DOCUMENT RESUME

ED 278 943 CS 008 639

AUTHOR	Schriver, Karen A.
TITLE	Revising Computer Documentation for Comprehension: Ten Exercises in Protocol-Aided Revision. CDC Technical Report No. 14.
INSTITUTION	Carnegie-Mellon Univ., Pittsburgh, PA. Communications Design Center.
SPONS AGENCY	Digital Equipment Corp., Maynard, Mass.
PUB DATE	Nov 84
NOTE	142p.; Portion of Chapter 4 may be marginally legible.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC06 Plus Postage.
DESCRIPTORS	*Computer Oriented Programs; Computer Uses in Education; Higher Education; *Protocol Analysis; Protocol Materials; *Reader Response; Reading Comprehension; *Revision (Written Composition); *Writing (Composition); *Writing Exercises
IDENTIFIERS	*Protocol Aided Revision

ABSTRACT

Based on the Protocol Aided Revision (PAR) system, this series of instructional units employs user protocols to provide the writer with feedback about a reader's response to a text and to help the writer become more sensitive to textual problems that may interfere with a reader's understanding. Geared to professionals who write on the job, the first of two introductory sections describes the text and includes both study strategies for maximizing understanding and advice on using the text. The second introductory section addresses teachers and provides information on the origin of the PAR system and outlines classroom activities. Each unit contains an example "problem text" in need of revision, a description of the user protocol based on the example, a summary of user questions/problems with the text, a revision of the problem text based on the reader's difficulties, strategies the writer used to revise the example, and a summary of strategies for revisors. Respectively, units cover (1) the control key, (2) commands for English text, (3) planning your document, (4) how to make a table of contents, (5) information commands, (6) introduction to "Mercury" (a mail system), (7) the Mail System, (8) introduction to "Oil" (a text editor), (9) the Integrated Library System, and (10) the design using system. (JD)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED278943

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

CDC Technical Report Series

CDC Technical Report No. 14

Revising Computer Documentation for Comprehension: Ten Exercises in Protocol-Aided Revision

Karen A. Schriver

November 1984

Communications Design Center

Carnegie-Mellon University
160 Baker Hall
Pittsburgh, PA 15213

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Karen A. Schriver

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

CS008639

BEST COPY AVAILABLE

The Communications Design Center was established in 1979 to solve communication problems in business, industry, government, and the professions.

The goals of the Center are:

- to encourage interdisciplinary approaches to the solution of visual and verbal communication problems;
- to promote cooperation between communication researchers at the University and those outside the University who have practical interests in communication problems;
- to provide resources and an organizational setting for research in visual and verbal communication;
- to train professional writers; researchers, and teachers;
- to promote the development of educational materials for communication programs on and off the CMU campus.

This report is one of a series from our on-going research on communication and document design. If you have questions or comments on this report, or if you would like to find out more about the activities of the CDC, contact:

Director
Communications Design Center
Baker Hall 160
Carnegie-Mellon University
Pittsburgh, PA 15213
(412) 578-2906

Revising Computer Documentation for Comprehension: Ten Exercises in Protocol-Aided Revision

Funded by the Digital Equipment Corporation

Written by:
Karen A. Schriver
Carnegie-Mellon University
Pittsburgh, PA 15213

Copyright (C) 1984 Karen A. Schriver

Table of Contents

Page

Introduction v-xii

- To Those Who Write on the Job v-viii
- To the Teacher ix-xii

Unit 1: The "CONTROL" Key 1-12

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 2: Commands for English Text 13-26

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 3: Planning Your Document 27-34

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 4: How to Make A Table of Contents 35-42

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 5: Information Commands 43-52

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 6: Introduction to Mercury 53-76

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 7: The MAIL System 77-92

- Example "problem" text in need of revision
- User protocol based on the example
- Summary of user questions/problems with the text
- Revision of the problem text based on the reader's difficulties
- Strategies the writer used to revise the example
- Summary of strategies for revisors

Unit 8: **Introduction to OIL** 93-108

Example "problem" text in need of revision
User protocol based on the example
Summary of user questions/problems with the text
Revision of the problem text based on the reader's difficulties
Strategies the writer used to revise the example
Summary of strategies for revisors

Unit 9: **ILS: The Integrated Library System** 109-122

Example "problem" text in need of revision
User protocol based on the example
Summary of user questions/problems with the text
Revision of the problem text based on the reader's difficulties
Strategies the writer used to revise the example
Summary of strategies for revisors

Unit 10: **The DUC System (Design Using Computers)** 123-134

Example "problem" text in need of revision
User protocol based on the example
Summary of user questions/problems with the text
Revision of the problem text based on the reader's difficulties
Strategies the writer used to revise the example
Summary of strategies for revisors

TO THOSE WHO WRITE ON THE JOB:

Our approach to writing technical documentation reflects a radical departure from tradition. In the traditional view, the way to improve the clarity of a document is to apply guidelines from style guides such as Elements of Style or The Practical Stylist, or to write to readability formulas such as those of Flesch[1] or Gunning.[2] While standard editing techniques often improve the effectiveness of a text, for example, by suggesting the use of active voice, high frequency words and simple sentence constructions, they are limited in the scope of text problems they help writers to see. Our research has shown that even the most skilled writers frequently fail to identify passages of text which seriously confuse readers. Further, writers' revisions sometimes actually make the text more difficult to understand.[3] These observations made it apparent to us that writers cannot rely strictly on these standard editing tools to produce consistently clear text. Rather, they need the uniquely clear window that readers' response to their text provides to tell them how well their prose is working.

This series of instructional units is based on a revision system we call Protocol Aided Revision (PAR). Protocol Aided Revision employs user protocols to provide the writer with feedback about the reader's response to a text. To obtain a user protocol, subjects are asked to think aloud while they read and try to use a text. When readers experience difficulty in comprehending the text, their comments reveal the location and, frequently, the nature of the difficulty. The protocol sessions are recorded on audio or video tape for analysis. Because they are collected while the person is reading and is engaged in the process of comprehension, user protocols provide much more explicit and complete diagnostic information than do readers' comments collected after reading is finished. Protocols can be used to revise texts in a sequence of revision cycles. In a revision cycle, we collect protocols from five subjects and then revise the text based on reader feedback. Typically, three or four cycles are sufficient to guarantee a clear text. We apply protocol methodology because we have seen repeated evidence that the reader's response to the text is the best available information on whether it meets the needs of the intended audience.

The objective of these instructional units is to make you more sensitive to the needs of audiences by using protocols to familiarize you with typical reader responses to complex texts.

---

[1] R. Flesch, The Art of Readable Writing, Harper and Brothers Publishers, New York, 1949 (revised Harper and Row, 1974).

[2] R. Gunning, The Technique of Clear Writing, McGraw-Hill Book Company, New York, 1952 (revised 1968).

[3] J. Swaney, C. Janik, S. Bond, and J. Hayes, "Editing for Comprehension: Improving the Process through Reading Protocols," Document Design Project, Technical Report No. 14, Pittsburgh, Pennsylvania, June 1981.

The ten units are designed to help you to become sensitive to a range of text problems that interfere with the reader's understanding. Our research indicates that writers trained in protocol aided revision are far better at predicting and anticipating possible reader problems in complex texts than are writers who are trained using standard methods of revision. Repeated exposure to readers' interaction with texts helps writers to diagnose more text problems and to diagnose those problems with greater specificity than do other techniques.

About this Text:

In each of the units to follow, your goal will be to predict and identify problems that a novice computer user may have with the example text--both before seeing a protocol and after seeing it. Each exercise consists of the following parts:

- an example "problem" text in need of revision
- a user protocol collected on the example text *Note: The protocol subject's comments are printed in boldface italics. When she/he simply reads the example text, no special indication is made.*
- a summary of the reader's questions/problems with the text
- a revision of the problem text based on the reader's difficulties with the text[4]
- an expert analysis of the revision strategies used to revise the text, strategies based on information obtained in the protocol
- a summary of revision heuristics that may be used by writers in similar revising situations

Suggested Study Strategies:

Each unit follows the above organization. To maximize your understanding of text problems that often confuse readers, we suggest you approach each unit as follows:

1. Read the "context" statement at the beginning of the unit. This will help to orient you to the type of audience for which the example text is intended as well as the particular context from which the text has been excerpted. You do not need to be familiar with the specific program or application that the content of the example text describes. Your goal is to use your experience as a technical writer to help you anticipate likely places in the text that may create reader confusion. You can assume the command sequences are technically accurate and that command sequences are not missing. You do not need expert knowledge of the computing system.
2. Read the example text and exercise your skill as a technical writer to evaluate and predict what features will cause problems for a naive user. During your reading, focus on any aspect of the text that may create confusion or misunderstanding on the part of a novice reader, that is, a person with little or no experience with computers. Each time you detect a potential reader problem, underline that portion of the text.

---

[4] The author wishes to thank the writers who helped in creating the revised texts: Sandra Bond, Carol Janik, John Hayes, Patricia Sullivan, Joyce Swaney, and Nina Wishbow. The creative efforts of these writers form an essential part of this text.

You may underline words, phrases, or whole sections, but try to mark each problem distinctly; that is, do not underline "as one section" places where you anticipate more than one problem.

3. Number each underline consecutively.

4. Examine each underline and think about what problem you anticipate that portion of the text will create for the reader. Take out some paper, and now for each numbered underline, briefly describe what the reader's problem will be. Number each description so that it corresponds to the number you assigned the problem in the example text. Characterize the problems as explicitly as possible, e.g., the fourth sentence is ambiguous because it lacks context, a novice reader will not understand this term, the second idea does not relate to the first, etc. When creating this initial list of potential reader problems, it is important that you examine only the example text and that you do not look ahead.

5. After you have generated your initial list of potential reader problems, put your list aside and take out another sheet of paper. Now turn next to the user protocol based on the example text with the goal of generating a second list of reader problems--problems you missed on your first pass. Read the protocol carefully, noting each time the reader responds to the text and ask yourself if the reader is having a problem. If the answer is yes, then underline and number the place where the reader has the problem try to characterize what you think that problem is. Write down only problems you missed during your initial reading.

6. Now compare your first and second lists and ask yourself, "what kinds of reader problems did I fail to predict in my initial list?" You will want to make special note of these problems so you will be sensitive to them the next time you see text that may trigger similar potential reader difficulties.

7. Next examine the list of reader questions that have been extracted from the protocol. Try to characterize these questions according to the reader problems they reflect, e.g., problems of content, of ambiguity, of context, of examples, of analogies, etc. We find this exercise will help you to consolidate your understanding of the reader's needs.

8. Now read the revised text. Pay attention to the types of revisions made and see if you can judge why the writer made them. Notice that in most cases, the content of the revised text is the exactly the same as the content of the example text. Where content has been altered, it has been deleted, not added. In this way, you will be able to see how the writer worked from the example text through the protocol to the revised text. For the most part, the example texts were technically accurate before revision was made. All revisions were generated after analyzing the reader's response during the protocol. You may disagree with the revisions made and might think of alternative successful revisions. In fact, we hope you will.

9. After you have examined the revised text, read through the strategies the writer used to revise the example. These strategies show, in a concise way, how the writers tried to resolve the problems they saw revealed in the protocol.

10. Last, read the strategies for revisors. These strategies are intended to provide you with general advice on typical problems readers often have with texts of this sort. We hope you will find them useful and applicable to other revising contexts.

11. Proceed in this manner through each unit.

Using This Text: Some Advice

- When predicting problems a novice audience may have, do not assume any technical background on the reader's part.

- Keep in mind that the protocols that follow the example text are actual recordings of a user trying to understand the text. If the reader has questions whose answers appear obvious, remember that the reader has no prior experience with the subject matter, and that questions were motivated by his/her effort to make sense of new information.

- Be an active reader when predicting and anticipating problems throughout these units. You will find the exercises most useful if you try to question the texts as a reader might.

- Read and adapt the guidelines presented at the end of each unit to your particular revising situation; we hope you will be able to use these revision strategies across a wide range of writing situations.

- Pay special attention to new problems you discover only after you read the protocols. These may be frequent text problems to which you need to be more sensitive. Determine whether the problems you diagnose after reading the protocols change, either in number or in kind, or both.

- We expect that you will get better at diagnosing text problems that confuse readers as you get more practice in seeing how people react to texts. After you complete about half of the units, calculate whether you are getting any better at predicting potential reader problems before you see the protocol. As you proceed, we anticipate that you will need to rely less and less on the protocols in predicting reader problems.

- When you finish all ten exercises, compare the kinds of problems you detect pre and post protocol. You may find that at the end of the exercises, your pre protocol diagnoses will be much improved (both in number and kind) over those you cited pre protocol during the early exercises. In fact, our aim in training you in protocols is to help you to improve your ability to predict and diagnose problems in texts without the need for examining a protocol.

We hope that you will argue with some of the issues we raise and think about alternative ways of conceptualizing readers' problems into guidelines for writers. If these units help you to think beyond standard editing and revision techniques, our goal will be met.

Karen A. Schriver, Writer and Faculty
John R. Hayes, Faculty and Director
Linda S. Flower, Faculty
Communications Design Center
Carnegie-Mellon University
Pittsburgh, PA

TO THE TEACHER:

This text grew out of my experience as a document designer, consultant, and teacher of technical and professional writing at Carnegie-Mellon University. It was funded by a grant from Digital Equipment Corporation.

Background:

Within Carnegie-Mellon's English Department, we have a unique component called Communications Design Center (CDC). The Communications Design Center is a multidisciplinary effort, drawing on the expertise of faculty and students in rhetoric, graphic design, and cognitive psychology. Beginning in 1979 with the Document Design Project funded by NIE, the CDC, along with Siegal and Gale Associates (based in New York) and the American Institute of Research (based in Washington, D.C.) launched a joint effort to produce basic and applied research into document design. Since that project, the CDC has been devoted to the planning, writing, testing, and evaluation of functional documents. Over the past few years, our students and faculty have worked on many types of documents, including computer manuals, leases, loan applications, informational brochures, insurance policies, and medical consent forms.

Through our research and our experience in document design, we have developed a new approach to writing technical documentation called Protocol Aided Revision (PAR). This approach reflects a radical departure from tradition. In the traditional view, the way to improve the clarity of a document is to apply guidelines from style guides such as Elements of Style or The Practical Stylist, or to write to readability formulas such as those of Flesch[1] or Gunning.[2] While standard editing techniques often improve the effectiveness of a text, for example, by suggesting the use of active voice, high frequency words and simple sentence constructions, they are limited in the scope of text problems they help writers to see. Our research has shown that even the most skilled writers frequently fail to identify passages of text which seriously confuse readers. Further, writers' revisions sometimes actually make the text more difficult to understand.[3] These observations made it apparent to us that writers cannot rely strictly on these standard editing tools to produce consistently clear text. Rather, they need the uniquely clear window that readers' response to their text provides to tell them how well their prose is working.

Protocol Aided Revision employs user protocols to provide the writer with feedback about the reader's response to a text. To obtain a user protocol, subjects are asked to think aloud while they read and try to use a text. When readers experience difficulty in comprehending the text, their comments reveal the location and, frequently, the nature of the difficulty. The protocol sessions are recorded on audio or video tape for analysis. Because they are collected while the person is reading and is engaged in the process of comprehension, user protocols provide much more explicit and complete diagnostic information than do readers' comments collected after reading is finished. Protocols can be used to revise successive drafts of documents. In a revision cycle, we collect protocols from five subjects and then revise the text based on reader feedback. Typically, three or four cycles are sufficient to guarantee a clear text. We apply protocol methodology because we have seen repeated evidence that the reader's response to a text provides the best available information on whether it meets the needs of the intended audience.

---

[1] R. Flesch, The Art of Readable Writing, Harper and Brothers Publishers, New York, 1949 (revised Harper and Row, 1974).

[2] R. Gunning, The Technique of Clear Writing, McGraw-Hill Book Company, New York, 1952 (revised 1968).

[3] J. Swaney, C. Janik, S. Bond, and J. Hayes, "Editing for Comprehension: Improving the Process through Reading Protocols," Document Design Project, Technical Report No. 14, Pittsburgh, Pennsylvania, June 1981.

The Origin of this Text:

The idea for this text was born of a need and a chance observation. The need was to provide a less expensive way to improve technical writing. Protocol Aided Revision is very effective but it is also costly. Some companies balk at the expense. I was searching, therefore, for a way to get some of the benefits of protocols without their costs. The chance observation was that many of the writers who used Protocol Aided Revision claimed that the experience had changed the way they wrote. They claimed that they were better able to understand the needs of their readers than they were before. This observation suggested that a text which captured and conveyed some of that direct experience with protocols could help writers to achieve a better understanding of their readers. This text is my effort to accomplish that goal.

Specifically, the purpose of these units is to help writers become more sensitive to text problems which interfere with the reader's understanding. We have organized each unit as follows:

- an example "problem" text in need of revision
- a user protocol collected on the example text *Note: The protocol subject's* ***comments*** *are* ***printed*** *in* ***boldface*** *italics. When she/he simply reads the* ***example*** *text, no special indication is made.*
- a summary of the reader's questions/problems with the text
- a revision of the problem text based on the reader's difficulties with the text
- an expert analysis of the revision strategies used to revise the text, strategies based on information obtained in the protocol
- a summary of revision heuristics that may be used by writers in similar revising situations

The units can be used alone (either in class or as self-paced instruction) or in conjunction with technical writing or other document design instruction. If the text is used as an integral part of a course, each unit can be (and is designed to be) used as a springboard for classroom discussion on problems readers have with texts.

I feel that it is important, when you first introduce the text, to emphasize to the students the importance of following the directions in the section "Suggested Study Strategies." I believe, for example, that if the students read the protocols before they attempt to diagnose the text problems on their own, that much of the beneficial effects of the protocols may be lost. Some classroom discussion of these instructions could help to avoid misunderstanding.

When discussing the protocols, remind students that some of the responses will be idiosyncratic. Some readers will object to an aspect of the text that would bother almost no one else. The student should not feel, therefore, that they must respond to every reader complaint. Writers must exercise their judgement to decide that the reader's complaint is serious enough to warrant revision. You should explain to your students that a reader's complaint in a protocol doesn't require them to revise, rather it's a flag which designates a place in the text where they should consider revision. If the same problem turns up in several protocols, that is strong evidence that the text is at fault.

At the end of each unit, the discussion of the expert revision summarizes the points that are most important for the student to attend to. I think that technical writers will gain a great deal from an exhaustive discussion of these points.

Suggested Classroom Activities:

- In each unit, the students are asked to read the example text in the unit and to list the problems they feel the intended user will have with the text. In my technical writing course, students needed about one half hour to complete a first pass on the "example text." It took them about twenty minutes to complete making their post protocol list. I found that assigning the "example text" as homework allowed me adequate time to have my students complete their second pass on the protocol in-class (50 minute period) and I still had half of the class time remaining for discussion. Notice that none of the example texts are longer than about seven pages and some are as short as one page. You may want to read through the units before choosing an order for presenting the units--the order is flexible. You will find is useful, especially in the early units, to encourage group discussion of the problem lists--both pre and post protocol.

- Ask students to present one reader problem cited on their first reading of the example text and one problem cited after reading the protocol. Put the problems on a blackboard, the first set on one side, the second set (post protocol) on the other.

- Ask students to compare the types of problems cited in the first versus the second (protocol) pass. Discuss: "Is there a difference in type? In specificity?"

- Give students the list of reader questions excerpted from the protocol text. See if the class can generate a coding or category scheme by asking students group similar kinds of questions. Ask them "given these problems categories, what kinds of revisions may be necessary?"
- Ask students to read the "revision based on the reader's difficulties with the text". Discuss the changes made and compare these with the student's suggestions for revision.
- Ask students to read and to discuss the strategies the writer used to revise the example.
- Ask students to read and to discuss the guidelines for revisors at the end of the unit. Discuss and relate to other problem texts.
- Ask students to keep a log of the most frequent reader problems they encounter.

I have tested this text empirically with 95 technical writers and find that writers who have worked through the manual show large and statistically significant advantages over other comparable writers in two ways: the trained writers diagnose more problems and they diagnose more kinds of problems in post-training tests than do writers who have not had this training. I am continuing the testing of the text to determine what it is that works best about the text and how best to continue to improve it. I welcome any comments you may have about how the text works with your students. Please send your comments or questions to me at the following address:

Karen A. Schriver
Instructor of Rhetoric
Department of English
Carnegie-Mellon University
Pittsburgh, PA 15213

UNIT 1

Context: The example text of Unit 1, "**The CTRL (Control) Key**" is part of an early draft of the Carnegie-Mellon TOPS-20 Manual, Volume 1. It occurs near the beginning of the manual and provides a tutorial on elementary keyboard operations. The primary audience for this tutorial was freshman college students enrolled in engineering, science, humanities or fine arts programs. No previous computer experience was assumed.

EXAMPLE TEXT:

The **CTRL** (Control) Key

The Control key allows you to use Control characters. Using this key involves two actions: Press the CTRL key down, and while keeping it pressed down, strike another character (usually a letter). When using the Control characters, it is not necessary to hit RETURN. However, it is necessary to keep the CTRL key pressed down while striking the Control character.

The following are the most commonly used CTRL characters:

CTRL C returns you to the EXEC (indicated by the @ prompt).

Two CTRL C's exits a program and returns you to the EXEC.

For example:

Action. Type *where* then hit RETURN.

Response. The computer begins typing a long list of information.

Action. Do two CTRL C's, that is, hold the CTRL key down the strike the C key twice.

Response. The program stops and the @ prompt appears.

CTRL U deletes the current line.

Suppose you have typed a rather long command,

and then decide that you don't want to issue that command. Instead of using the DELETE key to delete the command letter by letter, do a Control U.

Action. At the @ prompt, type: *information accounts.*

Action. Do a Control U.

Response. The command you typed disappears and the @ prompt reappears on your screen.

CTRL W deletes the current word or part of the command. Just like Control U, Control W is a more efficient way to delete text.

Action. At the @ prompt, type *information accounts.*

Action. Do a Control W.

Response. The word "accounts" disappears from your screen and the computer types @information.

CTRL R wipes out and then reprints the current line.

Suppose you are in the middle of typing a command and a system message appears. Do a Control R to reprint your command.

Action. Do a Control R.

Response. The computer wipes out and then reprints @information.

Action. Do a Control U.

Response. The computer deletes your command and the @ prompt appears on your screen.

CTRL S stops the flow of output on your terminal screen.

Suppose you are looking at a long list of information and you want to stop the list from scrolling up so that you can scan the list. In this case, you would do a Control S.

Action. Type *where* then hit RETURN.

Response. The computer types a list of

information.

Action. Do a Control S.

Response. The list of information stops.

CTRL Q resumes the flow of output on your terminal screen.

Action. Do a Control Q.

Response. The computer resumes typing the list of information.

Action. To return to the EXEC, do a Control C.

You've probably noticed a number of other unfamiliar keys, such as OFFLINE, BREAK, and BACKSPACE. Since you won't need to use these keys, we won't explain their function, except to note that the BACKSPACE key doesn't backspace. Instead, hitting BACKSPACE produces a series of ^H's. Don't use the BACKSPACE KEY to correct a typing error; use DELETE, CTRL U or CTRL W instead.

USER PROTOCOL OF EXAMPLE:

The control key. *O.K.* The control key allows you to do control characters. *I don't know what control characters are.* Using the key involves two actions. Press the control key down and while keeping pressed down, strike another character usually a letter. When using control characters. *It must be like return, delete, only they don't have their own key.* It is not necessary to hit return. *And it is necessary when your entering data? So I've got this kind of notion that you hit return when you've moved out of executive into entering data, and so then you have to hit return to get yourself back to, maybe back to control. And is control the same thing as being in the executive? Because you're controlling when you're in executive? Not necessarily...* The most commonly used CTRL characters. *The question I'm going to try to answer by reading this is, "What exactly is the difference between being in the executive and being in something else, and what are all these commands?"*

My theory now is that you have some commands that have keys and once you hit them you get out of control, or executive and you're doing something else, and return gets you back up to a command level, because there's a prompt, but that may not be true. O.k., well I have the practical side of this down, I think. Hold it down and hit it. *I'm not sure why.* Control key returns you to thee executive. *Oops, so much for my theory--* Indicated by the @prompt. In some cases you will need to do two controls. *For example, I'll do something. See what happens. O.k., I'm at the executive because I'm at the prompt. I type "where", o.k.; then I hit return. Oh, obviously not. An error has happened. O.k..* Begin typing a long list of information. Do two control C's. That is hold the CRTL key down and strike the C key twice. The program stops and then the @ prompt appears. *Oh, I should have been reading this while that was running. Yeah, I'm a bad subject. Hit return and then maybe keep reading. Let's see. Let me see if I can test myself. Where am I now? I'm obviously not in --let me see--If I type "where" it shouldn't work because I'm not in the command. So I need to get back to command. Let's see if I can do it with just one. Control C. No, that didn't help. Control CC. Oh, it worked! Now let's see if I can actually stop it. No, I thing I have to hit return and then it should start doing it again. o.k., control CC--great. O.k.,, stops and the prompt appears. O.k., so I'm getting in and out of executive.*

I need to know if that's the story line here, finding out how to move back and *forth between the two. So when I'm not in executive, it's not just that I'm entering data, it's doing something as well. O.k., so that's expanded language in that. That would really help me, I think, to have sort of story line at the beginning about what we're going to do is go in and out of executive, and your goal is to give commands. So that this kind of routing I'm going through would have some consistency. Does that make sense? I guess for me.*

Maybe if what I was told was that I'm going to be doing something to learn about these keys, and one of the things I'm doing--I can think of two ways to do it, maybe--is to say that you're going to be learning how to move from executive to whatever mode that is, and how to get back again. And then you might consider local goals, you know for these individual things where it seems useful. That's just a thought. But there's clearly an over-arching structure at the executive, because a lot of the keys and things you're telling me about are really--that's the unstated function, and I think I'm having to bring a lot of that, too. Although that's very motivating knowing that's that what I'm doing. It's kind of nice, other than just kind of doing this and seeing what happens, because I feel I have to memorize it by itself and it seems a little harder than memorizing it as--like an actual condition. If you want to do this, this is the key for you.

Control U, deletes current line. *O.k., now this time I'm goin to follow it through and I'm not going to play on my own.* Suppose you've typed a rather long command and then decide that you don't want to issue that command. Instead of using the Delete key ---*Right, terrific!* At the @prompt, type "information accounts." *Oh, you know what? I can barely stand this, you tell me to type "information accounts" but I know the technique. I'm going to do it and see if it works. It's like telling a little kid to make a graph. You know, just kind of for no reason or purpose at all. O.K., now do a control U. Oh, marvelous*

Control W deletes the current word or part of the command, just like Control U, it's more efficient way of deleting text. *o.k., information accounts.* Now do a Control W. *And that would be the current line? Control W. Oops, ah, so it deleted what I was supposed to be entering. That's what it did, parens was a word. The parens on the right and the parens on the left. Oh, that's very neat. One of the things I find myself not wanting to do is what you tell me, I'd rather play around and learn the thing. That's fun.*

Since the manual tells me what happens I don't actually have to do it! So I feel like I can learn and feel comfortable playing around. Control R, wipes out and reprints the current line. *Why?* Suppose you are in the middle of typing a command and a system message appears. Do a Control R to reprint your command. *Now that assumes if a system message appears I want to get rid of it. I don't understand. Either I want to get rid of that so it won't print out on my print out, or the system message has likely to have wiped out my command. This is confusing.*

Action. Do a Control R. Response. *Now wait a minute. Why would I--but I don't have any text. Well there's some here. Control R. Oh, this is so clever, they did plan where I was.* Action. Do a Control U. *We've already seen that* The computer deletes you command and the @prompt appears on your screen. *o.k., this is still complexing to me. So I'll just have to remember that if I ever get a system message, but I don't know why. I need to be told why. Oh, well, just scratch that key.*

Control S. Stops the flow of output on your terminal screen. *Now wait a minute! We already did that. Ah, you want to stop it from scrolling. o.k., so will it stopped so I can hit Control S?* Type "where" then hit return. *o.k., so it will stop so I can hit Control S. Wonderful. Now I'm goin to want to start again* Control Q. *Oh, there you are right there, just what I need. Control S. I want to get out of it. I think I'll go back and try a Control C. Control CC. There, it's all over with.* To return to the Exec do a Control Q and then a Control C. *Oh, I wouldn't have had to do CC. I could have done a single C. Interesting? Well that doesn't matter. I'll just do one or the other. I don't have to know why--or do I? Although I do recognize that it's an inconsistency in my knowledge. Apparently it's when you have long information and you want to get out of it, and the Q does the function of the first C. So it's QC or CC, maybe.*

You've probably noticed a number of other unfamiliar keys, such as Offline, Break, and Backspace. Since you won't need to use these keys, we won't explain their function. *Oh, that's good to tell me--except to know that the backspace key doesn't backspace. Oh wonderful. Yeah, I tried it and it didn't. Look at that* Produces a series of ^H's. Don't use the Backspace key to correct a typing error; use the Delete, Control U or the Control W instead.

I'm noticing the way I'm storing information--a single action: "deletion" through a series of steps. The organization around keys doesn't help me do that too much. It may be that there is no optimal organization for the text and that what you have to do is have it stored in different ways like in an index. It would be nice to have the commands organized, not just around the keys but around the functions. I'd like to get information instead of these awful things.

THE USER'S QUESTIONS WHILE READING THE EXAMPLE:

- What are the Control characters?
- When is it necessary to hit Return?
- Is being in Control the same as being in the Executive? What the difference?
- Why do you have to hold the Control key down?
- Does hitting a Control key return you to the Executive?
- Do I have to read the whole section before typing in my data?
- How do I move back and forth between the command and the executive level?
- What is the goal of this chapter? What are the local goals?
- Is there a story or narrative that you could tell me at the beginning that would help me to get oriented to the goal?
- What is the over-arching structure and purpose of these commands?
- Do I have to memorize these commands?
- Is there a difference between the purpose and the function of a command?
- What do I do if a system message appears? Does it wipe out my command?
- Could the command keys be organized around their functions?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

Using Special Keys

How to use the CTRL (Control) key

The CTRL key is always used in combination with another key, usually a letter key (such as c, r, u, etc.); such letter keys are called control characters. Control characters are used to give instruction to the computer. Using the CTRL key involves two actions:

1. Holding down the CTRL key.
2. Striking the control character.

Note that it is not necessary to hit the RETURN key after striking a control character. However, it is necessary to keep the CTRL key pressed down while striking the control character.

The purpose of the next four sections is to show you how to use six common control characters according to the function they perform.

1. How to get back to the Exec -- CTRL C

Now matter what task you're doing at your terminal, you can always get back to the Exec (i.e., back to the @ prompt) by doing a CTRL C. In some cases, you will need to do two CTRL C's. The following example uses the WHERE command to show how CTRL C works. (Note: some programs have special exit commands that you should use instead of CTRL C.)

Action 1. At the @ prompt, type *where* then hit RETURN.

> Response. The system begins typing a long list of information about who is logged in to the system, what program they are using, what terminal they're using, etc.

Action 2. Do two CTRL C's.

> Response. The system types ^C then the @ prompt appears on the next line, indicating that you are back at the EXEC.

2. How to Control Scrolling -- CTRL S and CTRL Q

Suppose in the previous example you wanted to stop the list as a certain point to read a particular line. You would do a CTRL S (for "stop"). If you wanted to continue looking at the list of information, you would do a CTRL Q to resume the flow of output on your screen. The following example uses the WHERE command to show how to control scrolling using CTRL Q and CTRL S.

Action 1. At the @ prompt type *where* then hit RETURN. Once the list begins scrolling, do a CTRL s.

Response. The list of information stops.

Action 2. Do a CTRL q.

Response. The list begins scrolling again.

Action 3. To return to the Exec, do a CTRL c.

3. How to reprint a line -- CTRL R

Suppose that you have typed a command and you notice some extra letters in you command. Since you're pretty sure you didn't type those letters, you would suspect that they are garbage characters; to make sure, you would do a CTRL r to reprint your command.

Action 1. At the @ prompt, type *information accounts* but don't hit return.

Action 2. Do a CTRL r.

Response. The system wipes out and then reprints your command on the same line.

Note: It isn't necessary to delete this command since we'll be using it in the next example.

4. How to delete -- DELETE, CTRL w and CTRL u

Suppose you have typed a command and you discover an error, or you decide that you don't want to issue the command. In this case you would need to delete all or part of the command by using one of three methods:

Action 1. Use the DELETE key to delete a few letters
in the word "accounts".

Action 2. Do a CTRL w.

Response. The rest of the word
"accounts" disappear.

Action 3. Do a CTRL u.

Response. The rest of the command
disappears and the @ prompt reappears
on you screen.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Changed the organization.
- Provided purpose statement.
- Consolidated similar information.
- Defined all unfamiliar terms.
- Provided short forecast of what to expect.
- Created action-oriented headings and subheadings; e.g., "How to..."
- Planned each example so the user could try each command with the practice text on the screen.
- Avoided unnecessary and confusing repetition.

STRATEGIES FOR REVISORS:

- The user needs to know what the goal of any function is. They need to know *why* to perform a function. Make the goal and function as explicit as possible.
- The user needs a context for the material presented. Provide a brief and concise context so the reader can see how the parts fit the whole.
- The user needs to know how to perform a function. Tell the user exactly what to do, and if there are any special features of a command, such as "holding down the Control key" make these features clear.
- The user needs to have unfamiliar terms defined and used consistently. This idea becomes critical when commands duplicate and overlap. They need to know which commands to use and when to use them.
- The user needs a clear understanding of the organization of the manual or section. They need to be able to conceptualize the governing structure of any set of related commands in order to visualize how they operate. This notion is also helpful for getting users to remember complex sequences of actions.
- The user needs reinforcement that he/she has performed the procedure correctly. Providing the user with the computer's "response" to any given action always helps reassure the inexperienced user that the action was indeed the correct one.
- The user needs *examples* of when he/she might want to perform a given command. When learning new commands, it is often difficult for new users to consider when they might want to use them.
- The user needs meaningful page or section headings. One of the most frustrating experiences for users is being unable to access information rapidly when needed. Make your headings explicit in terms of purpose and function.

UNIT 2

Context: The example text of Unit 2, "**Commands for English Text**" was taken from the EMACS Users Manual (Stallman, MIT, 1974). The Stallman manual provides a very detailed discussion of the EMACS editor. At one time it was the only readily available reference on EMACS. The original text was written for a general audience. The revision provided is written for novice computer users, e.g., freshman or secretarial staff at Carnegie-Mellon. No previous computer experience was assumed.

EXAMPLE TEXT:

Commands for English Text

Emacs enables you to manipulate words, sentences, or paragraphs of text. In addition, there are commands to fill text, and convert case.

Editing files of text in a human language ought to be done using Textmode rather than Fundamental mode. Invoke M-X Text Mode to enter Text mode. See section 20.1 [Major Modes], p. 85. M-X Text Mode causes Tab to run the function ^R Tab to Tab Stop, which allows you to set any tab stops with M-X Edit Tab Stops (See section 11.3 [Indentation], p. 46.). Features concerned with comments in programs are turned off except when explicitly invoked. Automatic display of parenthesis matching is turned off, which is what most people want. Finally, the syntax table is changed so that periods are not considered part of a word, while apostrophes, backspaces and underlines are.

If you are editing input for the text justifier TEX, you might want to use TEX mode instead of Text mode. See the file INFO:ETEX.INFO.

For Scribe input, use SCRIBE mode, which is like text mode but recognizes scribe comments and enables display of parenthesis matching. SCRIBE mode recognizes @BEGIN and @END lines as separating paragraphs but considers most other @-commands as ordinary text. If you have other lines which should separate paragraphs but do not start with @BEGIN or @END, put @; in front of them. For example,

@;@quotation[This is some text that should be filled by itself, not together with the preceding or following text.@;]

Someday there may be special major modes for other text justifiers.

Word Commands

Emacs has commands for moving over or operating on words. By convention, they are all Meta-characters.

M-F	Move Forward over a word.
M-B	Move Backward over a word.
M-D	Kill up to the end of a word.
M-Rubout	Kill back to the beginning of a word.
M-T	Transpose two words; drag a word forward or backward across other words.

Notice how these commands form a group that parallels the character based commands C-F, C-B, C-D, C-T, and Rubout.

The commands Meta-F (^R Forward Word) and Meta-B (^Backward Word) move forward and backward over words. They are thus analogous to Control-F and Control-B, which move over single characters. Like their Control-analogues, Meta-F and Meta-B move several words if given an argument. Meta-F with a negative argument moves backward like Meta-B, and Meta-B with a negative argument moves forward. Forward motion stops right after the last letter of the word, while backward motion stops right before the first letter.

It is easy to kill a word at a time. Meta-D (^R Forward Kill Word) kills the word after point. To be precise, it kills everything from point to the place Meta-F would move to. Thus, if point is in the middle of a word, only the part after point is killed. If some punctuation comes after point and before the next word, it is killed along with the word. If you wish to kill only the next word but not the punctuation, simply do Meta-F to get to the end, and kill the word backwards with Meta-Rubout. Meta-D takes arguments just like Meta-F.

Meta-Rubout(^R Backward Kill Word) kills the word before point. It kills everything from point back to where Meta-B would move to. If point is after the space in "FOO, BAR," "FOO," is killed. In such a situation, to avoid killing the comma and space, do a Meta-B and a Meta-D instead of a Meta-Rubout.

Meta-T(^R Transpose Words) moves the cursor forward over a word, dragging the word preceding or containing the cursor forward as well. A numeric argument serves as a repeat count. A negative argument undoes the effect of a positive argument; it drags the word behind the cursor backward over a word. An argument of zero, instead of doing nothing, transposes the word at point with the word at mark. For example, "FOO,BAR" transposes into "BAR FOO,".

To operate on the next n words with an operation which applies between point and mark, you can either set the mark at point and then move over the words, or you can use the command Meta-@ (^R Mark Word) which does not move point, but sets the mark where Meta-F would move to. It can be given arguments just like Meta-F. The case conversion operations have alternative forms that apply to words, since they are particularly useful that way.

Notice that if you are in Atom Word mode and in Lisp Mode, all the word commands regard an entire Lisp atom as a single word. See section 22.1 (Minor Modes), p. 107.

The word commands' understanding of syntax is completely controlled by the syntax table. Any character can, for example, be declared to be a word delimiter. See section 222.4 (Syntax), p. 111.

USER PROTOCOL OF EXAMPLE:

o.k., now I'm going to try Chapter Eleven: Commands for English Text. Emacs enables you to manipulate words, sentences, or paragraphs of text. *These commands sound like ones I'd use all the time--good.* In addition, there are commands to fill text, and convert case. *I don't know what it means to fill text, I guess it means putting data from one text into another... that is filling the text with what you want in it. Well, guess I'll soon find out.*

Editing files of text in a human language--*human language? boy that sounds strange, what could they be distinguishing there? Maybe computer language or machine language from human language?* ought to be done using Textmode rather than Fundamental mode. *Well, I don't know what textmode or fundamental mode is, so how will I know which I'm in? Let's see...* Invoke M-X Text Mode to enter Text mode. *I won't do that because I do not have time to see the other section. That's terrible to tell me to* See section 20.1 [Major Modes], p. 85. M-X Text Mode causes Tab to run the function ^R Tab to Tab Stop. *Now wait a minute, what's all this stuff? Sounds important, but I have no idea what it could mean. I know what tabs on a typewriter are, and I see a tab key here, but what the heck does running the function ^R Tab to Tab stop mean? Oh God, now they want me to look up another section* M-X Edit Tab Stops. See section 11.3 [Indentation], p. 46. *just to get started in this one. No I think I'll skip that. I'll deal with it when I have to use it. I thought this section was on how to edit; what does setting tab stops have to do with editing?* Features concerned with comments in programs are turned off except when explicitly invoked. *hmm, a feature? Wonder what a comment feature is? How do they get turned off? Do I have to turn them off? Well I guess I can't turn them off if I don't know what they are.* Automatic display of parenthesis matching is turned off, which is what most people want. *What does that mean? How do I know if I'm like most people and want them turned off? I wish they'd tell me why I need to know all these hacker terms!* Finally, the syntax table is changed so that periods are not considered part of a word, while apostrophes, backspaces and underlines are. *Syntax table...no idea. Of course, I do know what the syntax of sentences are, so maybe it means something to do with certain commands Emacs considers acceptable...maybe certain actions in a row produce a correct syntax.*

If you are editing input for the text justifier TEX, you might want to use TEX mode instead of

Text mode. See the file INFO:ETEX.INFO. *Well, I don't want to edit input for the text justifier mode, I just want to figure out how to manipulate words and paragraphs--like it says up here. This stuff is too complicated and it's aggravating to read.*

For Scribe input, use SCRIBE mode, *o.k., scribe I'ver heard of that, it formats the text on the xerox machine and it likes matching parentheses* which is like text mode but recognizes scribe comments and enables display of parenthesis matching. SCRIBE mode recognizes @BEGIN and @END lines as separating paragraphs but considers most other @-commands as ordinary text. *o.k., that I get.* If you have other lines which should separate paragraphs but do not start with @BEGIN or @END, put @; in front of them. For example, *oh good, an example.* @;@quotation[This is some text that should be filled by itself, not together with the preceding or following text.@;]

Someday there may be special major modes for other text justifiers. *Well so what! Who cares? What is the purpose of telling me that. Maybe somebody would be interested, but not me. I just want to try the darn commands, not get a preview of coming attractions.*

Word Commands. Emacs has commands for moving over or operating on words. By convention, they are all Meta-characters. *o.k., now what's a meta-character and what is meant by them being conventional? I don't see any key named "meta" I guess they aren't going to tell me. Great...They're conventional, but they're not on my keyboard. How do they expect me to try this out?*

M-F Move Forward over a word. *Well, I'll try the obvious, capital M, dash, capital F. Oh great, once again the obvious is not obvious. Now what? How about capital M without the dash, F. No, I just got capital MF...I could say something here about what I think about this command. And look there's a whole bloody string of them. O.k., how about control MF. No that didn't do it. How about if I hold down the M and the F at the same time, MF. No I got FM that time. O.k., I give up. No wait, let's try this button ESC. Oh no. Now I'm below the white line with an M and a dash. How do I get back? Hit the F stupid, F. Oh it worked, the cursor moved forward. Good. Why didn't they tell me that meta meant esc? O.k., I'll just zip through these next ones. Esc B. Why do I keep going below the white line? Maybe if I do it faster. ESC B. Right. ESC D. Now what is m-rubout? There's no rubout key. Not again. Yes folks, it's guess that key time again. Well it says* M-Rubout Kill back to the beginning of a word. *So that means get rid of as in delete. O.k., so escape delete, I'll try that. Good, I'm* *getting better at this. Now this next one.* M-T Transpose two words; drag a word forward or backward across other words. *That's easy...woops they flipped; guess that's right then.*

Notice how these easy commands *har har, no I don't notice that* form a group that parallels the character based commands C-F, C-B. C-D, C-T, and Rubout. *What is a character based command? No I don't get the analogy and I still don't know what the little @ signs do.*

The commands Meta-F (^R Forward Word) and Meta-B (^Backward Word) move forward and backward over words. They are thus analogous to Control-F and Control-B, which move over single characters. *O.k., now I get it: one set moves over chunks and the other set over letters--why didn't they say that earlier. I could have pictured this whole mess better if they had.* Like their Control-analogues, Meta-F and Meta-B move several words if given an argument. *I still don't know what an argument is.*

Meta-F with a negative argument *This is terrible, now negative arguments...beats me* moves backward like Meta-B, and Meta-B with a negative argument moves forward. Forward motion stops right after the last letter of the word, while backward motion stops right before the first letter.

It is easy to kill a word at a time. Meta-D (^R Forward Kill Word) *Wait, now what does ^ R mean? Is that some new command for meta-rubout or meta-delete?* kills the word after point. *Ah, what is the darn point? The space after the word to be deleted?* To be precise, *oh yeah, tell me about being precise! This is frustrating. Do I have to keep doing this?* it kills everything from point to the place Meta-F would move to. Thus, if point is in the middle of a word, only the part after point is killed. If some punctuation comes after point and before the next word, it is killed along with the word. If you wish to kill only the next word but not the punctuation, simply do Meta-F to get to the end, and kill the word backwards with Meta-Rubout. Meta-D takes arguments just like Meta-F. *o.k., at this point, I'll just take their word.*

Meta-Rubout (^R Backward Kill Word) kills the word before point. It kills everything from point back to where Meta-B would move to. If point is after the space in "FOO, BAR," "FOO," is killed. In such a situation, to avoid killing the comma and space, do a Meta-B and a Meta-D instead of a Meta-Rubout. *Why would I want to do that? I like examples, but this one is unclear and seems a little goofy. Why do I want to think about killing commas and spaces?*

Meta-T(^R Transpose Words) moves the cursor forward over a word, dragging the word preceding or containing the cursor forward as well. *Isn't that the same command I tried before? Why all these words?* A numeric argument serves as a repeat count. *Huh? What is meant by a numeric argument? I guess this stuff is for math majors or something. I don't want to know about numeric arguments anyway.* A negative argument undoes the effect of a positive argument; it drags the word behind the cursor backward over a word. An argument of zero, instead of doing nothing, transposes the word at point with the word at mark. For example, "FOO,BAR" transposes into "BAR FOO,". *Well I can't try it because I don't know what to do.*

To operate on the next n words with an operation which applies between point and mark, you can either set the mark at point and then move over the words, or you can use the command Meta-@ (^R Mark Word) which does not move point, but sets the mark where Meta-F would move to. It can be given arguments just like Meta-F. *I guess once I find out about arguments, this stuff might have some meaning. Now what?* The case conversion operations have alternative forms that apply to words, since they are particularly useful that way. *What are alternative forms of case conversion? The only thing I can think of is changing from capital to lower case lettering.*

Notice that if you are in Atom Word mode and in Lisp Mode, all the word commands regard an entire Lisp atom as a single word. See section 22.1 (Minor Modes), p. 107. *No I won't see section 22 because I won't know what I'm reading. Why do they keep sending me all over creation to find a simple answer. Maybe the answer isn't simple. I don't know what Atom Word mode or Lisp mode is and I don't think I'll be using them anyway.*

The word commands' understanding of syntax is completely controlled by the syntax table. Any character can, for example, be declared to be a word delimiter. See section 222.4 (Syntax), p. 111. *What is a word delimiter? Better yet, I still don't get the Syntax table. Well at least I know where to look it up if I ever need it. But since I don't know why or when I'd need it, I may never look it up.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- What does a fill text command do?
- What is Textmode versus Fundamental mode?
- Why do I have to look up another section just to get started in this one?
- What does running the function ^R Tab to Tab stop mean?
- What does setting tab stops have to do with editing them?
- What is a comment feature? How does it get turned off? How do I explicitly invoke them?
- What does automatic display of parenthesis matching mean? How do I know that's what I want?
- What is a syntax table?
- What is the text justifier? Do I need to know that?
- What's the difference between Scribe mode and Text mode?
- What is the purpose of telling me that someday there may be special modes for other text justifiers?
- What is a character based command? I don't get the analogy.
- What is a meta-character?
- What does the ^ mean? Is it the same as meta?
- What is an argument? What is a negative argument?
- Where is the meta-Rubout command on my keyboard?
- What is a point?
- Is meta-rubout the same as meta-delete?
- What is meant by "a numeric argument serves as a repeat count"?
- What is Atom word and Lisp mode?
- What is a word delimiter in a syntax table?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

Emacs Meta Commands for Editing Prose

Do I Want to Use this Tutorial?

Emacs is a useful editor if you write articles, letters, reports or any other blocks of prose. Emacs helps you to avoid endless retyping because it can transpose words, lines, paragraphs, chapters, or even complete articles. The purpose of this tutorial is to teach you new emacs commands that perform the following:

A. **Words:** How to move the cursor right or left one word or how to kill text right or left one word and how to transpose words.

B. **Sentences:** How to move backwards and forwards between sentences.

C. **Paragraphs:** How to move over or between paragraphs.

D. **Indentation:** How to indent you text.

E. **Tab Stops:** How to set up tab stops. This command is especially helpful in designing tables, charts, or anything that needs to have columns.

F. **Text Filling:** How to fill a text, that is how to fill-in gaps after you have deleted sections in the middle of a paragraph or region.

G. **Case Conversion:** How to convert a text from upper to lower case or vice versa.

H. **Dissociated Press:** How to scramble a text so that no one can read it. This command is especially helpful for playing practical jokes on friends, but on the more serious side, it is helpful for disguising your work when big brother is watching.

This chapter assumes that you have used Emacs to edit your text and are familiar with the basic commands found in *Introduction to CMU TOPS 20, Volume One*. We will cover the above commands in a similar manner as the Tops 20 manual, so if you have been reasonably successful with Emacs up to this point, you may find this chapter helpful if you need to perform any of the commands (A-H) above.

You should note that most of the following commands make use of the ESC key. ESC key commands are often called "Meta" commands. The ESC key is much like the CTRL key but it works on words and sentences rather than individual letters or numbers. To use the ESC key, strike ESC once (but do not hold it down as you would the CTRL key) and then strike the letter that goes with the command.

A. Word Commands

This section will show you how Emacs allows you to move the cursor over words, to kill individual words either forwards or backwards or even just part of a word, or to transpose words.

1. Moving the cursor right or left one word

- Moving forward:

  Action Strike ESC key and then hit f (forward).

  Response. The cursor moves to the beginning of the next word.

- Moving backward:

  Action Hit ESC key and then hit b (backward).

  Response. The cursor moves back to the beginning of the previous word.

2. Killing a Word Forward, Backward, or Simply Part of a Word

- To kill a word forward:

  Action Move the cursor to the first letter of the word and hit ESC d (delete).

  Response. The word disappears.

- To kill a word backward: (that is the previous word)

  Action Move the cursor to the last letter of the word and hit ESC and then hit the Delete Key.

  Response. The entire word disappears.

- To kill part of a word:

  Action Move the cursor to the first letter of the part of the word you want deleted and hit ESC and then delete.

For example: If you move the cursor to the "a" in Transylvania:

Response. The word is deleted from the "T" to the first "a", and you will have "nia" left on your screen.

3. Transposing Words

- To transpose two words:

Action Move the cursor to the beginning of the second word and hit ESC T (transpose). For example, to transpose the phrase "off blow" to "blow off" move the cursor to the "b" in the word "blow," and strike ESC T.

Response. The two words are transposed; e.g. off blow----> blow off

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Provided a table of contents. You will notice the table of contents includes more topics than those discussed in the brief example. The topics included were found in the same chapter and were discussed in the same meandering and wordy style.
- Told the user why they would want to use this tutorial; e.g., to move words and paragraphs.
- Explained assumptions about the level of sophistication expected to use this chapter; e.g., familiar with basic commands in Tops 20 manual.
- Made judgements about what reader questions were important and organized the section to address only those.
- Contrasted Meta key with Control Key to provide a more concrete analogy.
- Removed references to other sections of the manual.
- Used action-response tutorial format.
- Collapsed the "^," "M," and "Meta" to " ESC"--the actual acronym on keyboard.

STRATEGIES FOR REVISORS:

- The reader needs to know what prior knowledge is expected if they are to use the text most efficiently and effectively. If the text content builds on other information, define the source. If the content leads to more sophisticated information, provide the user with a long range goal. This strategy helps readers to see where they have been, where they are, and where they are going.

- The user needs a clear introduction to sections where commands can be subsumed under a general category. Provide short descriptions that distinguish purposes of commands within the context of the general category.

- The user needs to know in what contexts the text is most useful. Provide the reader with typical user situations.

- The user needs to be able to see the connections between the command and the keystroke(s) involved in invoking the command. For instance, in this unit, even though terms such as "meta" are more technically precise, using ESC better serves the reader's needs because it anchors the command with a physical referent on the keyboard.

- The user needs analogies for commands/ systems that have parallel functions. Providing the user with concrete analogies helps them to understand key relationships between concepts, ideas, and structures. Presenting clear likenesses between commands/ systems helps the reader to access the most relevant background information and helps them to filter out irrelevant and perhaps misleading expectations.

UNIT 3

Context: The example text of Unit 3, "**Planning Your Document**" is part of an early draft of the SCRIBE text-formatting program. The revision is included in the second volume of the Carnegie-Mellon TOPS-20 Manual. Consequently, readers are assumed to be unfamiliar with SCRIBE, but to have experience with other aspects of the TOPS-20 system such as using EMACS. The primary audience for this tutorial was freshman college students enrolled in engineering, science, humanities or fine arts programs--students who need to use scribe to format papers, reports, resumes, etc. No previous SCRIBE experience was assumed.

EXAMPLE TEXT:

Planning Your Document

At this point we assume that you have some text that you want to SCRIBE. Your text may already be in a file in your directory, or you may have a handwritten or typed copy. Choosing the appropriate SCRIBE commands to put in you manuscript file involves some preliminary planning and decision-making. Before you go on, consider the following questions:

- Is your text a rough draft that you'll need to revise a number of times, or is it ready to be produced as a finished document?
- Do you want your text to be single-spaced, double-spaced, etc.?
- Does your paper or report have a number of headings and subheadings?
- If so, how many levels of headings are there? In other words, if you were to construct a hierarchy from your headings and subheadings, how many levels would there be in your hierarchy?
- Do you want your headings and subheadings to be numbered so that they reflect the structure of your hierarchy?
- Do you need a title page and a Table of Contents?
- Do you need an index?
- Do you have any footnotes?
- Will you need blank spaces (or blank pages) in order to insert figures (such as graphs, charts, tables)?

- Will you be citing references and therefore need a bibliography at the end of your paper?

Once you have an idea about how you want your text to be formatted, you're ready to go on to the next section and choose the SCRIBE formatting commands that will produce the results you want.

USER PROTOCOL OF EXAMPLE:

At this point we assume that you have some text that you want to SCRIBE. Your text may already be in a file in your directory, or you may have a handwritten or typed copy. Choosing the appropriate SCRIBE commands to put in you manuscript file involves some preliminary planning and decision-making. Before you go on, consider the following questions: Is your text a rough draft that you'll need to revise a number of times, or is it ready to be produced as a finished document? *Now I assume what that question wants me to think about is am I really ready to do this or should I think about what I want to say before I begin to do it? On the other hand, perhaps it would be best to scribe my text for a draft just to see what it will look like on the page, mostly because I can't visualize how it will look.*

Do you want your text to be single-spaced, double-spaced, etc.? *I wonder if the spacing you choose depends on the font family you're in? I wish there was a way to see how the text would look when finished, then I would know whether I wanted it single or double spaced. I wonder if there is a command to display how it will look? It doesn't say here.* Does your paper or report have a number of headings and subheadings? *I'd like to know how these sections would look if I chose to use them.*

If so, how many levels of headings are there? In other words, if you were to construct a hierarchy from your headings and subheadings, how many levels would there be in your hierarchy? *Now I assume that's something about setting your commands beforehand? But I'm not sure why...I guess there's no limit on the number of levels one can have.* Do you want your headings and subheadings to be numbered so that they reflect the structure of your hierarchy? *I can think of examples where I'd want to letter my headings, but not number them.*

Do you need a title page and a Table of Contents? *It seems to me that the questions above would dictate the answer to this. If you wanted a bunch of levels, I guess you'd need a table of contents.* Do you need an index? *If I did, I'm sure I couldn't think of how to organize it...* Do you have any footnotes? *I wonder if there are ways to get footnotes either at the bottom of the page or at the end of a paper?* Will you need blank spaces (or blank pages) in order to insert figures (such as graphs, charts, tables)?

Will you be citing references and therefore need a bibliography at the end of your paper? *Now I suppose all of these things are not simply questions you should ask yourself, but* *decisions that SCRIBE requires of you when you start. So perhaps maybe it would be good for the reader to see it in two stages: one, don't go to SCRIBE until you really have a text to type in; two, SCRIBE is going to ask you questions about the following information so you ought to have the information and know what you want to do with respect to these questions.*

Once you have an idea about how you want your text to be formatted, you're ready to go on to the next section and choose the SCRIBE formatting commands that will produce the results you want. *That doesn't make a lot of sense. If I knew how I wanted it formatted, I wouldn't be reading this. Maybe they should say once you know what you want, you can pick the commands.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE:

- Does this section assume I should know the answers to all these questions before starting?
- Does spacing depend on font family?
- Can I see how the document will look before I produce a final version?
- Is there a limit on the number of levels I can have?
- What are the questions I need to consider under the notion of a table of contents?
- Are these questions I must know how to answer before going to Scribe?
- Can I get footnotes at the bottom of the page or at the end of a paper?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

PLANNING YOUR DOCUMENT

In order to use SCRIBE, you'll need a text that you want SCRIBE to format. Your text may be a draft that you'll need to revise, or it may be a final version that you're ready to produce as a finished document. In addition, your text may already be in a file in your directory, or you may have a handwritten or typed copy.

Once you have a text, you'll need to make some decisions about how you want your text formatted in order to choose what SCRIBE commands to put in your manuscript file.

Before you go on, consider the following questions:

- What kind of output device do you want to view you document file on?

  In most cases you'll want to print your document file on one of the available printers, such as the Xerox 9700 or a line printer. In some cases you might want to type your document file out on your terminal screen, which is another kind of output device.

- What type of document have you written--a letter, book, journal article, report?

- Do you want you text to be single-spaced or double-spaced? Do you want indented paragraphs?

- Do you need a title page? Do you need a Table of Contents?

  If you want SCRIBE to produce a Table of Contents, you'll need to think about the organization of your text. For example, how many major sections are there? How many subsections? How many levels of sections are there? (By "levels of sections" we mean that if you were to create a hierarchy of the section headings in your document, how many levels would there be in the hierarchy? For example, the Table of Contents for this manual has three levels.)

- Do you have any footnotes? (See section 6 for formatting options)

- Do you need an index?

- Do you need blank spaces on certain pages in order to paste in figures or graphs? Or do you have charts or tables that you want SCRIBE to format?

- Do you want to cite references in the text and include a bibliography at the end? (See section 6 for details and options)

Once you've decided what your formatting requirements are, you're ready to go on and choose the SCRIBE commands that will produce the results you want.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Defined assumptions needed to use section.
- Answered the reader's initial questions by clarifying points made in the introductory paragraph.
- Provided hypothetical answers to key questions.
- Cross-referenced answers to reader questions that were relevant to other more detailed sections.
- Changed order of the itemized questions and provided answers to those considered essential.
- Added information on viewing your document and the options available.
- Subsumed related questions on producing a Table of Contents.
- Changed background and summary advice.

STRATEGIES FOR REVISORS:

- Novice users need pointers regarding how to think about planning documents. When providing such information, you can make planning easier by anticipating their likely questions and by grouping those that fit together.

- When generating questions for the user to consider, it is difficult to select those that will be relevant in all circumstances. It is wise, therefore, to present the most general questions and to assume that readers will infer from the general question's answers to specific questions pertaining to their particular task.

- In this unit, the example text asks readers to direct their attention to decisions they need to consider. But since readers face a diversity of tasks, we cannot prescribe the outcome of their decisions. We can only remind them of certain critical decisions they must make and suggest strategies that will help them to make wise decisions. For instance, although the reader of the example text was to decide how to choose appropriate Scribe commands, he could not visualize a formatted document before producing a final version. Note how the revision of the example text starts with a reminder about the several ways to view the document.

- If content in the text serves different functions, it is often useful to use text format to reflect those differences. Further, the user should be able to scan the page and see the text's structure. In the revision, notice how the answers are separated from the questions by a change in typeface.

- Users will often ask questions whose answers are found in other sections. If a section is intended as a general preview, you can avoid answering specific questions by cross-referencing. However, too many cross-references frequently irritate users and make the general text appear unnecessary. Try to keep cross-references down to a minimum, roughly no more than 2-3 to a page.

- Users need to know what to consider at the end of one section to enable them to start the next section. Summarize these considerations so they tie in with the purpose of the section(s) that follow. In this unit, for example, once the user decides on the formatting requirements, starting the next section on Scribe commands can be done without much difficulty. Also, the user can selectively attend to only those commands that meet his particular requirements.

UNIT 4

Context: The example text of Unit 3, "**How to Make a Table of Contents**" is part of an early draft of the SCRIBE text-formatting program. The revision is included in the second volume of the Carnegie-Mellon TOPS-20 Manual. Consequently, readers are assumed to be unfamiliar with SCRIBE, but to have experience with other aspects of the TOPS-20 system such as using EMACS. The primary audience for this tutorial was freshman college students enrolled in engineering, science, humanities or fine arts programs--students who need to use scribe to format papers, reports, resumes, etc. No previous SCRIBE experience was assumed.

EXAMPLE TEXT:

How to get a Table of Contents

In order for SCRIBE to produce a Table of Contents and numbered headings, you must select one of the sectioned document types--article, report, or manual--and then use the appropriate section formatting environments. The table below shows the environments you can use for each sectioned document type and the type of numbering SCRIBE produces.

Level	Numbering	Document Type:	
		Article	Report and Manual
1	1.	@section	@chapter
2	1.1	@subsection	@section
3	1.1.1	@paragraph	@subsection
4	1.1.1.1		@paragraph
1	A.	@appendix	@appendix
2	A.1	@appendixsection	@appendixsection

When you use sectioning environments, you don't necessarily have to think of a section of your document as a chapter, for example, or a paragraph. Rather, you goal is to reflect the hierarchy of the parts of your document. Each of the environments listed above requires text--the title of each section--as an argument.

When you run your manuscript file through SCRIBE, SCRIBE automatically numbers the sections and produces a Table of Contents.

USER PROTOCOL OF EXAMPLE:

How to get a Table of Contents. In order for SCRIBE to produce a Table of Contents and numbered headings, you must select one of the sectioned document types--article, report, or manual--and then use the appropriate section formatting environments. The table below shows the environments you can use for each sectioned document type and the type of numbering SCRIBE produces. *Level 1, the section of an article is called a chapter. No wait, an article and a report have different formats. Why? What do they look like? A section is for an article and a chapter is for a report or manual. Oh all right, two is called a subsection and three 1.1.1 is a paragraph. O.k., you don't get four levels with an article. But you do with a report.... so the difference is the addition of a chapter in the front of a report or manual . 1A gives you an appendix. Oh yea, level 1 A gives you an appendix. Oh no, that's label A with a period. And to get and appendixsection, I would have it titled A.1. And to get one level of an appendixsection, I would just type in level 2.*

When you use sectioning environments, you don't necessarily have to think of a section of your document as a chapter, for example, or a paragraph. Rather, you goal is to reflect the hierarchy of the parts of your document. Each of the environments listed above requires text--the title of each section--as an argument. *O.K., an argument? What does that mean? Is my text the argument or is the section heading the argument? Now, let me see what we're doing here. When do I type those things in? And which of these commands go where? I wonder if I have to use all of them for any article or manual?*

When you run your manuscript file through SCRIBE, SCRIBE automatically numbers the sections and produces a Table of Contents. *I'm not sure what I'm doing here. This chart doesn't help much, I'm mixed up. How does the chart relate to my punching keys? I don't know what to type. Do I type @section? And where do I type that?* You must select one of the section document types, and then use the appropriate section formatting environment. The table below shows the environments you can use for each sectioned document type and the type of numbering SCRIBE produces. *Now this is not a global command, this is a local command. So it must be that before each chapter heading I have to say "chapter." I think. Does this refer to the section on hierarchies? Do you have to know where those hierarchies are going to come? You have to sort of have your document all figured out before you begin to pick levels.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE:

- What is the difference between how an article and a report get formatted?
- What do these formats look like?
- What is an argument?
- When and where do these commands get typed in?
- Do these commands refer to the section on hierarchies?
- Do I have to use all of these commands for either type of document?
- Do you have to figure all this stuff out beforehand?
- How does the chart connect to the actions I must perform?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

How to get a Table of Contents

In order for SCRIBE to produce a Table of Contents, you must select on of the sectioned document types--article, report, or manual--and then use SCRIBE's sectioning environments in your manuscript file. Each sectioning environment takes the text of the section heading as an argument. The table below lists the names of SCRIBE's sectioning environments, the document type you may use them in, and the numbering scheme SCRIBE uses to number each section.

SCRIBE'S SECTIONING ENVIRONMENTS

Document Type:		Numbering Scheme:
Article (3 levels)	Report and Manual (4 levels)	
@section	@chapter	1.
@subsection	@section	1.1
@paragraph	@subsection	1.1.1
	@paragraph	1.1.1.1
@appendix	@appendix	A.
@appendixsection	@appendixsection	A.1

Each time you begin a new section in your manuscript file, you use the appropriate sectioning environment with the section heading as an argument.

When you use sectioning environments, you don't have to think of a section of your document as a chapter, or a paragraph. Rather, your goal is to reflect the hierarchy of the parts of your document. The next example of a manuscript file shows how to use sectioning environments. To conserve space, we've indicated in italics where the actual text of the sections would appear.

Manuscript file:

```
@device<x9700>
@make<manual>

     other global commands

@chapter<Using the CMU Computer System>
@section<Reference Guide to CMU Facilities>
@subsection<System Information>

     text of this section

@subsection<Computer Accounts>

     text of this section
```

Document file:

1. USING THE CMU COMPUTER SYSTEM

1.1. REFERENCE GUIDE TO CMU FACILITIES

1.1.1. SYSTEM INFORMATION

text of this section

1.1.2. COMPUTER ACCOUNTS

text of this section

When you run your manuscript file through SCRIBE, SCRIBE automatically adds numbers to the sections according to the numbering scheme in the preceding table. The appearance of the headings depends on the global commands you use in your manuscript file. Notice that in the example above all of the headings are in boldface caps--even though we didn't use caps in our manuscript file--and the chapter heading is in a larger font (known as display font).

Scribe keeps track to the sectioning commands you include in your manuscript file and uses that information to produce a Table of Contents like this:

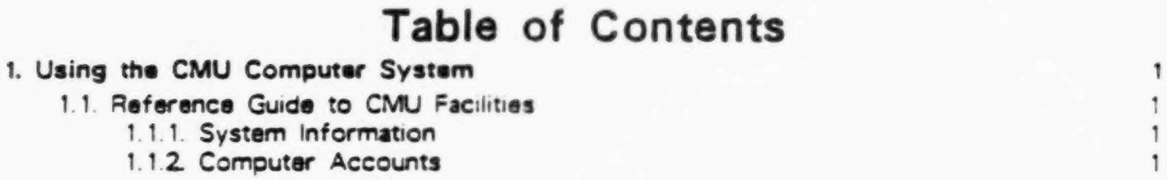


Table of Contents

1. Using the CMU Computer System 1
  1.1. Reference Guide to CMU Facilities 1
    1.1.1. System Information 1
    1.1.2. Computer Accounts 1

When SCRIBE creates a Table of Contents, it puts the actual page at the end of the document file; when you pick up your printout, you'll need to move the Table of Contents up to the front of your document.

If you print your document file on the Xerox 9700 with the duplex option--duplex means printing on both sides of the page--your Table of Contents may be printed on the back side of the last page. To correct this problem you could edit your manuscript file and add a blank page at the very end in order to force the Table of Contents page on to the next new page.

Note: For more information about how to correct this problem, at the Exec type
help scribe-duplex and hit RETURN.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Expanded the unit to make a clear distinction between command options and where to invoke commands within a sample text.

- Reformatted the chart to make it is easier to follow: subsumed the notion of levels under appropriate document types, eliminated the extra column, and provided a title for the chart.

- Provided an extended example on how to use the sectioning environments through including a sample manuscript file, a document file, and a Table of Contents.

- Explained how the numbering scheme in Scribe works.

- Pointed out the most important features of the sectioning environments to attend to.

STRATEGIES FOR REVISORS:

- When presenting charts that summarize information, take time to unpack its elements by "walking the user through" an illustration. Notice how in this unit, we chose the manual--the more common document type requiring a hierarchy of levels--to show the user how it looks when sectioning environments are executed.

- Most users want to access information quickly, but even when they have the time to peruse your document, purposeful formatting of the charts you present is critical. Provide meaningful and specific headings for charts and make certain you use enough white space to accent its features. You may want to use a different type face style to make it stand out from the text.

- If the difference between a manuscript file and a document file is part of the point of your section, display it both ways. Note how we moved from examples of the manuscript file to the document file to the Table of Contents. When you give the user examples that are concrete enough to make comparisons, you reduce the risk of major misunderstandings.

- When describing commands that ask users to make choices, make clear what actions to invoke and when. In this unit, the protocol revealed that the user did not know when to invoke commands. We solved this problem by moving from general to more specific information. Consequently, expert users will not have to read past the chart. On the other hand, for novice users, there is ample information to meet their needs.

UNIT 5

Context: The example text of Unit 5, "**Information Commands**" is part of an early draft of a tutorial on system commands; it is included at the end of the first volume of the Carnegie-Mellon TOPS-20 Manual. Consequently, readers are assumed to have a genral familiarity with the TOPS system, but not with the information commands. Since most Carnegie-Mellon users have several accounts on the system, accounts with varying allocations for printing, computing, and storage space, these commands are important for all users.

EXAMPLE TEXT:

Information Commands

To Get Information on the Amount of Computer Space Allocated to You and How Much You've Used

Action. At the @ prompt type *i dis* (for information on disk usage and hit RETURN key.

Response. The first line of the response tells you how many pages you are using. The second line tells you how many pages you have been allocated. The third line tells how many pages are free and how many are being used on the whole system. An example of this response is:

```
PS:  <jh12>
 397 Pages assigned, 195 in use, 193 deleted
 800 Working pages, 400 Permanent pages allowed
 29427 Pages free on PS:  373413 Pages used.
```

In the above example, user JH12 is using 397 pages. Of those, 195 are in files which will be kept at logout, and 193 are in files which have been deleted and which will be expunged upon logout or use the "expunge" command. The user has been allotted 800 working pages (to handle "save" files, deleted but not expunged files, etc.), but can have only 400 pages at logout. The last line gives an example of how busy the system is (there is generally no problem unless there are 500 or fewer pages free).

In all cases, a page consists of 512 36-bit words.

To Get Information about your Directory

Action. At the @ prompt type *i dir* then hit RETURN.

Response. Here is an example of the message which appears:

```
Name PS<JH12>
 Working disk storage page limit 800
 Permanent disk storage page limit 400
 Number of directory 1051
 Default file protection 777700
 Account default for LOGIN N146
 Protection of directory 77752
 Last LOGIN 9-Mar-82  08:47:47
 User groups 260436 (JH12)
 Directory groups 325601 (LG01)
```

Storage space is explained above under "information on disk-usage" command. Protection codes are explained in Appendix A. The account default lists the account the computer time will be charged to if the user receives a computer account. The user group and directory group are assigned when the user receives a computer account.

USER PROTOCOL OF EXAMPLE:

Information Commands. To get information on the amount of computer space allocated to you and how much you've used, at the @ prompt type *i dis* and hit RETURN key. *Two hundred and ninety thousand pages used? Wow! I guess that's what it says. It also says I've got almost 420 permanent pages allowed. Eighteen--ah, none of these numbers add up. Eighteen hundred and forty working pages. I don't know what that means. Four hundred twenty permanent pages allowed. It must mean I can leave them there for the next thirty years? Thirteen thousand nine hundred and eighty eight pages free. It doesn't tell me anything that I can understand. Do I really have to know all this stuff?*

The first line of the response tells you how many pages you are using. *hmm... the first line, eh?* The second line tells you how many pages you have been allocated. *This doesn't match that. Well I don't know, I'll go on.* The third line tells how many pages are free and how many are being used on the whole system. *Pages free? I don't get that.* In the above example, user JH12 is using 397 pages. Of those, 195 are in files which will be kept at logout, and 193 are in files which have been deleted. *Well let's see what that means about me. JH-12 is using 397 pages. I'm using 110 pages. Of those, 195 are in files in use. Mine doesn't match this one. So I have no unuse and no deleted? I guess so. Now the second line.* 800 working pages. *What does that mean?* The user has been allotted 800 working pages to handle "save" files, deleted but not expunged files, but can have only 400 pages at logout. *I'm not sure I know what that means either.* The last line gives an example of how busy the system is. *O.k., I really don't understand that one. And this stuff about "there is generally no problem unless there are 500 or fewer pages free" is a big waste for me. How can I know what the problem is? Does it mean free for me? Free for my port group or the whole darn system?* . In all cases, a page consists of 512 36-bit words. *What's a "bit-word"? Oh, skip it.*

To Get Information about your Directory at the "at" prompt type i dir then hit return. *O.k, working disk storage page limit 1840, permanent 420...number of directory. Well, again my numbers don't match the example. Gee, aren't computers manuals fun? It's a shame to have such a piece of equipment in front of me with a manual I can't make heads or tails of.* Storage space is explained above under "information on disk-usage" command. Protection codes are explained in Appendix A. *Oh great, why not here when I need it? At least stuff on disk usage...Wait, I just read info on disk-*

usage! But I still don't get the storage space idea. And I still don't know how to find out what I have left--and that I thought was the purpose of this section. The account default lists the account the computer time will be charged to if the user receives a computer account. *I'm going to quit and skip the rest of this madness. I can't distinguish any of this page limit junk. Why do the numbers keep changing every time I try the command? Can't they collapse these ideas in some other way? It's really more information than I can understand. It would be helpful to know how to ask for information on disk-usage and on my directory and, more important, when I would need to know these things. At the moment, I just don't have the foggiest.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- What do all these huge numbers mean?
- Are they supposed to "add up"?
- Do permanent pages mean I can leave them there for the next 30 years?
- Do I really have to know all this stuff?
- Shouldn't my numbers match the example?
- What does working pages mean?
- What does the log out amount of pages mean?
- What does "how busy the system is" mean?
- When will I need to know these limits?
- Can't this stuff by collapsed in some other way?
- What's a bit word?
- Why do the numbers keep changing every time I try these commands?
- How do I find out how much space is left in my directory?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

How to Get Information About Your Space and Money Allocation

To find out how much file storage space has been allocated to you and how much you are using, or how much money has been allocated to your computer account and how much has been spent, use the *information* command with *disk-usage* or *account* as the argument.

To get information on file storage space:

Action. At the Exec type *i dis* (for "disk-usage") and hit RETURN.

Response. The response will be in this form:

```
PS:<jh12>
397 Pages assigned, 195 in use, 193 deleted
800 Working pages, 400 Permanent pages allowed
29427 Pages free on PS:  373413 Pages used.
```

The first line tells how much space your files currently occupy. These numbers change as you work.

Pages assigned is the total number of computer pages used to store all of your files (including files deleted but not yet expunged and temporary files in the buffer). (In computer terms, a page consists of 512 36-bit words. In more familiar terms, 1 computer page equals about 500 words of text).

In use (which may or may not appear when you use this command, depending on what tasks you have been performing) refers to the number of computer pages being used to store all of the files that will be retained at logout (that is, all of your files except deleted files and temporary files in the buffer).

Deleted (which may or may not appear when you use this command) refers to the number of computer pages that you have deleted and which will be subtracted from Pages assigned when either an expunge or logout occurs.

The second line tells how much space you are permitted to use to store files and to perform tasks. This is the same information obtained when you issue the *information directory* command. These numbers stay the same unless you get a new allocation.

Working pages is the total number of computer pages you are allowed to use **while you are logged in**, as explained in the *information directory* command section.

Permanent pages allowed is the number of pages you are permitted to have when you log out, as explained in the *information directory* command section.

The third line tells how much space is free on the system and how much space is being used by everyone with a directory on the system to store files and to perform tasks. These numbers constantly change as people work.

Pages free on PS is the number of computer pages left on the public space to handle all users' files. When the number of pages is down to 1,000, the system will warn that space is low. If there are 500 or fewer pages free, you will not be able to create or edit any files. All you will be able to do is read existing files.

Pages used is the number of computer pages being used to store all users' files and to handle tasks being performed by all users.

To find out how much space you have left: issue the expunge command to get rid of deleted files; next type *i dis* and hit RETURN; then subtract "Pages assigned" on the first line from "Permanent pages allowed" on the second line.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Provided a brief introduction that presents the purpose of the tutorial.
- Told the user that information on disk-usage can be found in either of 2 ways and gave the necessary arguments.
- Reformatted the entire text and subsumed information about each concept from the example under outdented headings.
- Underlined and started each sentence with important information.
- Defined the term "bit" in a way that relates to the user's purpose.
- Expanded each "page limit" concept in simple language, yet provided enough detail to distinguish concepts.
- Told the reader which numbers constantly change.
- Provided a conclusion and told the reader how to find out how much space is left.

STRATEGIES FOR REVISORS:

- Users often have difficulty understanding texts that contain new information embedded in every sentence. When revising texts such as this unit's example, consider ways to separate embedded information by changing the format. Extract the text's key features and provide this information first.

- Users look for ways to chunk information they read, and texts should reflect obvious ways to consolidate what is read. Headings, captions, indentations, and other formatting devices help the reader to see a text's structure. Notice how the revised example presents information in an obvious and easy-to-summarize way. Buried information from the example text is made prominent through the strategy of subsuming key information under outdented headings; e.g., "the first line tells you..." Further, terms are highlighted with underlining.

- When novice users read examples provided in a manual that do not match what they see on their terminal screen, they are often mislead and feel they have made a mistake. This unit's user protocol reveals this type of problem: The user did not understand why the disk-usage numbers did not match those on the screen. Further, the user did not know why the numbers kept changing as he continued to work. The user needs to be told when not to expect a one-to-one mapping between the example text and what is seen after evoking a command. This idea becomes most important when a command's function is to give the user "update" information.

- Users sometimes forget information they have just read; consequently, they fail to draw inferences between concepts presented in a text. This unit's user protocol illustrates this difficulty: even after completing the section on "getting information about space and money allocation," the user could not remember having read about disk-usage and could not infer the relationship between the concepts of "pages assigned" and "permanent pages allowed". The user never solved the problem, but rather gave up. One way to reinforce key ideas in a text is through functional redundancy, or explaining an idea several times in different ways. Notice how the revision of the example text makes use of functional redundancy: it first explains the concepts separately and then in the final statement, it makes the process of "getting information about space" simple and direct.

UNIT 6

Context: The example text of Unit 6, "**Introduction to Mercury**" describes a mail system, "Mercury"; it is part of an advanced programming environment being developed at Carnegie-Mellon. The text is taken from an internal document written by the program developer. At the time it was written, it was the only document describing "Mercury". Therefore, one might expect that the intended audience is a general one. The revision is written as a tutorial for a mixed audience--one that was described to the revisor as consisting of secretaries and engineering and computer science professors. The revision comes as the fourth chapter in the user's manual, consequently the particular system and its environments are previewed in earlier chapters. The revision therefore includes only a brief explanation of the system's function.

EXAMPLE TEXT:

1. Introduction to Mercury

This is a general guide to the syntax for Mercury commands and a few bits of advice for Mercury users. Read this section before trying to send messages in Mercury. All commands are printed in **boldface**.

Mercury Syntax

The SYNTAX Command **Sy Command-Name** will provide you with the syntax of a command. Basic features of Mercury syntax are given below:

- The command separator is the semicolon (;). Use the semicolon if you write more than one command on a line - **Mercury-Command;Mercury-Command.**
- If you wish to perform a monitor command, use an exclamation point (!), and the remainder of the commands line is passed to the shell to be executed - ! arbitrary-text.
- To separate names in a list use a comma (,). The comma is the list argument separator.
- For arguments using set expressions, you must use quoted strings. A quoted string is formed from matching single or double quotes, a dollar sign or a vertical bar.
- A sharp sign (#) is used to mark a comment. **# arbitrary-text** signals text which Mercury will not try to interpret.

A Few Words of Advice Before Starting:

- Send bug reports to Mercury@CMU-CS-G. Use the BUG Command by typing [**bu** at the prompt.

- Be aware that unprotected files can be read by other users.
- Don't run Mercury from two places on the same message file. You will damage the file.
- Usually, if no argument is given following a command, Mercury will assume you mean the current message or message file.

II. Getting Started, Message Attributes, Basic Commands.

Creating a File for the First Time.

To get Mercury to create your message file in your HOME directory, either type **Hg** at the monitor prompt or, if that doesn't work, type **/use/local/bin/hg** at the monitor level. Unlike the Greek god, this Mercury can be slow to start up, but eventually you will be placed in Mercury, which you will know by the prompt, **hg>** extension in your directory. After establishing a Mercury file, use the **Hg** Command to enter Mercury.

Whenever you enter Mercury, it will check to see if you have gotten any new messages, and will print the headers of the new messages.

Message Format: Headers and Bodies

The header of a message is a summary of its contents. A header has six fields:

1. The NUMBER of the message, assigned according to when it was received.
2. "Flags" which tell you about ATTRIBUTES of the message. The flags are:

   + the message is NEW.
   ? " " has not been ANSWERED.
   - " " has not been EXAMINED.
   * " " has been DELETED.
   B You sent a BLIND message - you saved a copy of the message in you mailbox *not yet implemented*.
   D You are composing a DRAFT of the message.

3. The DATE in the message was sent.
4. Who the message was FROM.
5. The SUBJECT of the message.
6. The NUMBER of characters in the message.

To read the headers of your messages, type **H Message-sequence**. The body of the message is the main text.

Creating and Sending Messages.

The commands to send messages are MAIL, FORWARD, ANSWER and REMAIL. We will discuss the MAIL Command and sending commands first. At the end of this section you will read about the minor differences of using the FORWARD, ANSWER and REMAIL commands.

To create a message, you will usually use the MAIL command. At the prompt, type **M** mail-name-list. Mercury will create a new message field for you, providing header field in the beginning of the message environment. Remember to separate multiple recipient names with commas. Use of user ID's is discouraged. A mail name is followed by the network cite of the receiver, separated by "at" or "@" such as:

To: John Smith@CMU-CS-S,Jean Doe@CMU-CS-G

Header prompts are fairly easy to interpret. The only prompt that may be puzzling is "CC" - which stands for carbon copy. Types the names of the people you wish to send copies of the message in this field.

Skip a line between message header and body.

Mercury's default editor, OIL, allows you to edit your message. When you leave the editor, an action prompt at the bottom of the screen allows you to execute on of several commands:

- MAIL - This is the usual response, which queues the message for delivery.
- EDIT - Allows you to edit the message; returns you to the Action prompt when done.
- BLIND/NO BLIND - Blind places a copy of the message in your mailbox and leaves you at the Action prompt for further instructions. NOBLIND cancels an automatic BLIND profile option.
- ABORT - Returns you to the monitor and EXPUNGES your text unless you use BLIND to save a copy.
- DRAFT - Returns you to the monitor without sending the message. Use the RETRY Command to finish a DRAFT. You cannot DRAFT an ANSWER.
- TYPE - Displays the current text on the screen.
- RETRY - Resumes composition of the DRAFTEd messages.

You can use the TYPE and REVIEW commands at the HG> prompt as well as while you are composing a message. Type: **Command Message-sequence**.

The other ways to compose a message are with the ANSWER, FORWARD, and REMAIL Commands.

Answer. **A Message-Sequence**. Used for replying to a list of messages. The default **To** in the header is a list of all messages you are answering.

Forward. **F Message-sequence** allows you to send copies of messages.

REMAIL. **REM Message sequence List-of-mail-names** is the fast form of FORWARD, sending messages in the left-hand argument to the receivers in the right-hand argument.

Basic Commands for Manipulating files and Leaving Mercury

If you want to get rid of messages, use the DELETE command: D Message-sequence. If you want the message back, type UNDELETE message-sequence. If you are sure you will never need the message again, use the EXPUNGE command; but remember that Expunging a message makes it impossible to UNDELETE it. Mercury will show you the headers of the expunged messages, and will ask for confirmation before expunging them. KILL is a variant of DELETE. Typing K-message sequence will DELETE the message and present the next message in the file on your screen.

Leaving Mercury

The EXIT command will return you to the monitor and will EXPUNGE all deleted files. You must reply "abort" to prevent expunging a deleted file. The word "no" is not accepted at the query. Typing QUIT will return you to the monitor and leave the DELETED files in your directory.

Basic Commands.

All Mercury commands manipulate the *current* message. The current message is the last message to be brought to Mercury's attention. The Type Command, ty, displays messages. The Next Command, n, will cause the message following the current message to appear. The Previous Command, p, displays the message preceding the current one.

If you want to get rid of messages, use the Delete Command: d. The message will remain in your file until you exit Mercury, at which point it will be expunged. If you change your mind about deleting a message, use the Undelete Command, und. The Expunge Command; exp, permanently removes deleted messages. Mercury will ask for confirmation before expunging messages. The Kill command, k, is the same as a Delete Command followed by a Next Command.

To issue a command that will act on other messages besides the current one, you type additional information after the command. This additional information is known as a command argument. For now, the only argument you need is the number of a message. Mercury assigns a number to a message according to the order in which Mercury received it. To find the number of a message type the Display-All-Headers Command, h all. Mercury will print out short descriptions of the messages you have on file. These short descriptions are known as headers. The first field of the short header is the message number. Try this command before reading on.

Once you know the number of a message, you can give message numbers as command arguments. For example, if you want to delete the third message in your file, type d 3. Or, to display the second message in your file, type ty 2.

A final note on message numbers. After a message has been expunged from your message file, the numbers of the remaining messages change. So, for instance, if you expunge the third message in your file, message 4 will become message 3, message 5 will become message 4, etc.

Reading Mail and Searching through Message Files.

Commands for Reading Mail. When you invoke Mercury, it checks the mailbox for new messages and displays their headers. While in Mercury, you must execute a Checkmail Command, ch, to see if any new messages have arrived. If you have received new mail, use the Type Command, ty to display it. Use the Type Command followed by the number of a message to display any other message. If you have forgotten the number of the message, type h all to get a list of the short headers of your messages. The first field of each header is the number of the message.

USER PROTOCOL OF EXAMPLE:

Introduction to Mercury. This is a general guide to the syntax for Mercury commands and a few bits of advice for Mercury users. Read this section before trying to send messages in Mercury. *Before trying to send messages? Well, I guess it's a program for sending messages, but I'm not sure. I don't even know if that's what the program is all about or if that's a special application...I'm just not sure.* All commands are printed in boldface.

Mercury Syntax. The SYNTAX Command Sy Command-Name will provide you with the syntax of a command. *I guess I type in everything I see in bold print.* Basic features of Mercury syntax are given below:

The command separator is the semicolon (;). *I'd like to know what is being separated, an example would clarify things for me.* Use the semicolon if you write more than one command on a line - Mercury-Command;Mercury-Command. *They said all commands are given in boldface, but I wonder if that's literal. I bet I don't just type Mercury-Command. I bet that's arbitrary. I don't know, I'll just keep going.* If you wish to perform a monitor command, *What is a monitor command?* use an exclamation point (!), and the remainder of the commands line is passed to the shell to be executed - ! arbitrary-text. *What is the shell? Anything that I've seen in bold so far isn't helpful. I wouldn't know what to type.*

To separate names in a list use a comma (,). The comma is the list argument separator. *Oh boy, well. When would I want to use a list? OK, keep reading. But I still can't do anything. For arguments using set expressions, you must use quoted strings. What are set-expressions? A quoted string is formed from matching single or double quotes, a dollar sign or a vertical bar. How do I match quotes? A sharp sign (#* is used to mark a comment. *OK, I've seen comments before. If I hadn't, I wouldn't know what's up.* **# arbitrary-text** signals text which Mercury will not try to interpret. *o.k., sounds like programming or something.*

A Few Words of Advice Before Starting. Send bug reports to Mercury@CMU-CS-G. Use the BUG Command by typing **Bu** at the prompt. *OK...* Be aware that unprotected files can be read by other users. *I don't know how to set protection.* Don't run Mercury from two places on the same message file. You will damage the file. *OK...How would I actually do that? Does it take two people? Do they mean simultaneously or do they really mean once you* *have Mercury or a message at one terminal you do it again from another terminal? Probably not.* Usually, if no argument is given following a command, Mercury will assume you mean the current message or message file. *huh? I don't know what they mean by arguments, and I don't know what they mean by current. This is advice to think about before starting? I'm afraid to start! Great...*

II. Getting Started. *Now that I don't know what to do before getting started, I'll start.* Message Attributes, Basic Commands.

Creating a File for the First Time. *You know I heard that Mercury was a mail program, but if I hadn't heard that, I would never know by reading this.*

To get Mercury to create your message file in your HOME directory. *What's a HOME directory? I guess it's just to create a message file.* Either type **Hg** at the monitor prompt or, if that doesn't work, type **/user/local/bin/hg** at the monitor level. *How do I know when I'm at the monitor level? I wonder if just typing those will work OK, or will I have to hit RETURN or something the way you do on the TOPS system. OK, I'm going to assume that you would have to hit return after all of those.* Unlike the Greek god, this Mercury can be slow to start up. *Oh I hate computer humor* but eventually you will be placed in Mercury, which you will know by the prompt, **hg>** extension in your directory. *You will know Mercury by the prompt? I guess your prompt changes, OK. I'm confused. I thought a prompt was on the screen like HG> and an extension was something added to a filename like junk.hg.* After establishing a Mercury file, use the **Hg** Command to enter Mercury. *This sounds like lost in space. I guess what they are telling me is hit hg and hit return. I still want to know how to establish the HG file.*

Whenever you enter Mercury, it will check to see if you have gotten any new messages, and will print the headers of the new messages. Message Format: Headers and Bodies. The header of a message is a summary of its contents. A header has six fields. *I'm still not clear on what a header is and I'm not so sure I understand fields, and I'm not going to bother with memorizing these flags--that seems to be the only way I'd remember them. I don't know why I need to know them now anyway. An example would help here. Enough of this.*

Message Format: Headers and Bodies

The header of a message is a summary of its contents. A header has six fields: *I'd like to know what this header looks like.*

1. The NUMBER of the message, assigned according to when it was received.

2. "Flags" which tell you about ATTRIBUTES of the message. The flags are:

   + the message is NEW.
   ? " " has not been ANSWERED.
   - " " has not been EXAMINED.
   * " " has been DELETED.
   B You sent a BLIND message - you saved a copy of the message in you mailbox *Are these messages you receive, messages you send or both?*
   D You are composing a DRAFT of the message.*Why do you need this message? Don't you already know if you are composing a draft?*

3. The DATE in the message was sent.

4. Who the message was FROM.

5. The SUBJECT of the message.

6. The NUMBER of characters in the message.

To read the headers of your messages, type **H Message-sequence**. The body of the message is the main text. *What is the main text?*

Creating and Sending Messages. The commands to send messages are MAIL, FORWARD, ANSWER and REMAIL. We will discuss the MAIL Command and sending commands first. At the end of this section you will read about the minor differences of using the FORWARD, ANSWER and REMAIL commands. *If I'm not going to learn about these commands yet, why am I told about them?*

To create a message, you will usually use the MAIL command. At the prompt, type **M** mail-name-list. *I assume that--wait, I'm not sure if I type M mail- Joe, Bill, Sam or I type Mail--Joe, Bill, Sam. I would guess the first?* Mercury will create a new message field for you, providing header field in the beginning of the message environment. *An example would be nice here.* Remember to separate multiple recipient names with commas. Use of user ID's is discouraged. A mail name is followed by the network cite of the receiver, separated by "at" or "@" such as:

To: John Smith@CMU-CS-S,Jean Doe@CMU-CS-G

Header prompts are fairly easy to interpret. The only prompt that may be puzzling is "CC" - which stands for carbon copy. *Confusing...why is "CC" puzzling?* Types the names of the people you wish to send copies of the message in this field. *Does this refer to carbon copies?*

Skip a line between message header and body.

Mercury's default editor, OIL, allows you to edit your message. · When you leave the editor, an action prompt at the bottom of the screen allows you to execute one of several commands: *I think I get it. After I write a message I can do any one of these things with it.*

- MAIL - This is the usual response, which queues the message for delivery.*Do I usually create the message in OIL or HG?*
- EDIT - Allows you to edit the message; returns you to the Action prompt when done. *What is the Action prompt?*
- BLIND/NO BLIND - Blind places a copy of the message in your mailbox and leaves you at the Action prompt for further instructions. NOBLIND cancels an automatic BLIND profile option. *"Blind" seem like a funny mane for this action. What is a profile option?*
- ABORT - Returns you to the monitor and EXPUNGES your text unless you use BLIND to save a copy.
- DRAFT - Returns you to the monitor without sending the message. Use the RETRY Command to finish a DRAFT. You cannot DRAFT an ANSWER. *What is an answer?*
- TYPE - Displays the current text on the screen.
- RETRY - Resumes composition of the drafted messages.

You can use the TYPE and REVIEW commands at the HG> prompt as well as while you are composing a message. *What is the review command?* Type: **Command** Message-sequence. *Is this what I actually type or does message-sequence stand for something else?*

The other ways to compose a message are with the ANSWER, FORWARD, and REMAIL Commands.

Answer. **A Message-Sequence**. Used for replying to a list of messages. The default **To** in the header is a list of all messages you are answering. *Oh, here's "Answer", I assume "A" is the command for answer.*

Forward. **F Message-sequence** allows you to send copies of messages. *Don't I have to say who I want to send copies to?*

REMAIL. **REM Message sequence List-of-mail-names** is the fast form of FORWARD, sending messages in the left-hand argument to the receivers in the right-hand argument. *I don't understand the difference between Forward and Remail. Why is remail capitalized? Is it just a mistake?*

Basic Commands for Manipulating files and Leaving Mercury *I don't get it, how do you have files in a mail system? I thought it was just messages.*

If you want to get rid of messages, use the DELETE command: D Message-sequence. If you want the message back, type UNDELETE message-sequence. *Oops, I thought message-sequence meant the message. Here it seems to mean the message name. What do message names look like?* If you are sure you will never need the message again, use the EXPUNGE command; but remember that Expunging a message makes it impossible to UNDELETE it. Mercury will show you the headers of the expunged messages, and will ask for confirmation before expunging them. KILL is a variant of DELETE. Typing K-message sequence will DELETE the message and present the next message in the file on your screen. *OK, I've heard of these types of commands before. Will "K" do for Kill? Is the "K-" a mistake?*

Leaving Mercury *That's good to know about. But here?*

The EXIT command will return you to the monitor and will EXPUNGE all deleted files. You must reply "abort" to prevent expunging a deleted file. *Reply to what?* The word "no" is not accepted at the query. *At what query? Do we need an example?* Typing QUIT will return you to the monitor and leave the DELETED files in your directory.

Basic Commands. All Mercury commands manipulate the *current* message. The current message is the last message to be brought to Mercury's attention. The Type Command, ty, displays messages. The Next Command, n, will cause the message following the current message to appear. The Previous Command, p, displays the message preceding the current one.

If you want to get rid of messages, use the Delete Command: d. The message will remain in your file until you exit Mercury, at which point it will be expunged. If you change your mind about deleting a message, use the Undelete Command, und. The Expunge Command; exp, permanently removes deleted messages. Mercury will ask for confirmation before expunging messages. The Kill command, k, is the same as a Delete Command followed by a Next Command. *This is a lot to remember, maybe a list would help.*

To issue a command that will act on other messages besides the current one, you type additional information after the command. This additional information is known as a command argument. For now, the only argument you need is the number of a message. Mercury assigns a number to a message according to the order in which Mercury received it. To find the number of a message type the Display-All-Headers Command, `h all`. Mercury will print out short descriptions of the messages you have on file. These short descriptions are known as headers. The first field of the short header is the message number. Try this command before reading on. *An example would clarify this whether I was on line or not.*

Once you know the number of a message, you can give message numbers as command arguments. For example, if you want to delete the third message in your file, type d 3. Or, to display the second message in your file, type ty 2.

A final note on message numbers. After a message has been expunged from your message file, the numbers of the remaining messages change. So, for instance, if you expunge the third message in your file, message 4 will become message 3, message 5 will become message 4, etc.

Reading Commands *That sounds ambiguous. I assume that these are commands for reading you mail.*

When you invoke Mercury, it checks the mailbox for new messages and displays their headers. While in Mercury, you must execute a Checkmail Command, ch, to see if any new messages have arrived. If you have received new mail, use the Type Command, ty to display it. Use the Type Command followed by the number of a message to display any other message. If you have forgotten the number of the message, type h all to get a list of the short headers of your messages. The first field of each header is the number of the message. *What's the difference between a mailbox and a mailfile? This is a lot to learn and it's very confusing.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- What is Mercury? What is its purpose?
- Is it important to send messages, or is that just a side-feature of the system?
- Do I type everything I see in boldface?
- Should I type these commands exactly as I see them in the manual?
- How should I interpret the term "mercury-command?"
- How should I interpret the command descriptions? I don't understand the explanations for executing commands.
- What is a list argument separator?
- What is a monitor command?
- What is the shell?
- When would I want to use a list?
- What are set expressions?
- How do I match quotes?
- How do you run Mercury from two places at once?
- How do I know when I'm at the monitor level?
- What does a header look like?
- What is the main text?
- How do you set file protection?
- What is an argument?
- What does the author mean by "the current message?"
- What does the "advice" in the beginning of the text mean?
- Why didn't the author include a description of what Mercury is?
- What is a home directory?
- How do you execute the command? Do you hit RETURN?
- Why do computer people like to make obscure jokes?

- Does the cursor prompt change when you invoke Mercury?
- Do I have to know information about fields now?
- Do I usually create the message in OIL or in HG?
- What is a profile option?
- Why is information about leaving Mercury presented here?
- What is an ANSWER? How does HG know when I am answering a message?
- What is the REVIEW command?
- Why aren't examples provided?
- What's the difference between FORWARD and REMAIL?
- Will typing "K" do for "killing" something?
- What's the difference between a mailfile and a mailbox?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

Chapter Four
Mercury

4.1 Introduction

Mercury is Accent's system for sending and receiving mail. Mercury was designed to allow you to send, receive, and store many messages, and to create more than one file for storing messages. Therefore, you can use Mercury in the same way you would use any other mail system, and you can also use Mercury as a "file cabinet" for organizing and retrieving many correspondences.

Mercury works by removing new messages from a mailbox file and placing them into a message file. You give Mercury various commands when you want to read and organize your messages and when you want to create and send new messages. Mercury lets you create multiple message files and define classes of messages to ease searching through long files.

Please read these words of advice before starting:

To execute the commands given in this chapter, type the italicized abbreviations. In other words, if the instructions tell you to type the Delete Command, *delete*, type the word, "delete."

Mercury commands usually take arguments--these will be described. If no argument is given, Mercury acts on the current message (the last message to be brought to Mercury's attention).

Don't run Mercury from two terminals on the same message file. You will damage the file.

You must run Mercury to execute any of the commands in this chapter (with the exception of invoking Mercury from the monitor).

4.2 Getting Started

4.2.1 Invoking Mercury

To invoke Mercury	*hg* RETURN When Mercury starts up for the first time, it creates a file with the extension .hg in your directory and gives you the prompt Hg>

4.2.2 Leaving Mercury

To exit Mercury and expunge all deleted messages	*ex* (for "exit") RETURN Mercury will ask for confirmation before expunging any messages.
To prevent expunging deleted files	*q* (for "quit") RETURN

4.2.3 Getting Help

If you need information, use the Help Commands. Remember that you must first invoke Mercury before you can execute any command given in this chapter.

- To get a list of Mercury commands: *?* RETURN
- To get a description of a command: *help* *<command name>* RETURN
- To see the correct syntax for a command: *syntax* *<command name>* RETURN

4.3 Sending and Reading Mail

4.3.1 Sending Messages

To create a message, use the Mail Command: *m* RETURN

Mercury will create a skeleton message environment for you. Mercury's default editor is Oil. At the top of the message environment is a header field. Header prompts tell you where to insert: the subject of your message (this is the `Re:` field), the date, receivers of carbon copies (the `CC` field of the message), and the people to whom the message is addressed (the `To:` field). The `CC` field is optional; the `To` field must be completed. To complete the `To:` field, type the receiver's name, an @ sign, and their network site.

If you want to practice using Mercury, try sending a message to yourself. To do this, type your own user name and network cite in the `To:` field. To send messages to more than one person, separate multiple recipient addresses with commas, as shown in the example below:

`To:` *John Smith@CMU-CS-S,Jean Doe@CMU-CS-G*

If you want to send copies of the message to anyone, address the `CC` field in the same way as the `To` field.

A line of text shows where to begin the body of the message. Start typing at this prompt line. You may delete the prompt line--this will not affect your message, but you must skip a line between the message headers and the body of the message.

When you finish typing the message, exit the editor with CTRL X F An Action prompt will appear at the bottom of the screen. The Action prompt allows you to decide what to do next with the message you have written and provides the abbreviation for each command in capital letters. Type the Action prompt command you wish to select, and hit RETURN. The prompts are:

ABort	Returns you to the `hg>` prompt and deletes your text.
Blind	Places a copy of the message in your message file and leaves you at the Action prompt for further instructions.
Draft	Returns you to the `hg>` prompt without sending the message; places a Draft copy of the message in your message file.
Edit	Returns you to the editor.
Mail	The usual response, which queues the message for delivery.
Type	Displays the current text on the screen.

There are other ways to create messages, but if you are using Mercury for the first time it is best to use the Mail command. If you know how to give command arguments using message attributes and keywords, skip the rest of this section and go on to section 4.3.2.

Other ways to compose a message are with the Answer, Forward, Retry and Remail Commands. Remember to hit RETURN after you type in these commands.

a	The Answer Command, followed by a list of messages to be answered, is used for replying to a list of messages. Separate multiple messages with commas. The `To:` line in the header will contain the list of messages you are answering. The subject field in the header is `Re:` followed by the subject of each message you are answering.
f	The Forward Command, followed by a list of messages to be fowarded, a slash, (/), and a list of recipients, sends copies of the messages to the people in the list. You may add explanatory text to the beginning of Fowarded messages.
rem	The Remail Command, followed by a list of messages, a slash (/), and a list of recipients, is the fast form of Forward. The Remail Command does not send you to the editor; it automatically sends the messages in the left-hand argument to the receivers in the right-hand argument.
ret	The Retry Command, followed by a draft message, resumes composition of the message.

4.3.2 Basic Commands for Reading Mail and Manipulating Messages

All Mercury commands manipulate the current message. The current message is the last message to be brought to Mercury's attention. The commands listed below are the ones you will use most often to read mail.

t	The Type Command displays messages.
n	The Next Command causes the message following the current message to appear if there is one.
p	The Previous Command displays the message preceding the current one.

The following commands are ways to get rid of messages.

d	The Delete Command deletes messages. The message will remain in your file until you exit Mercury, at which point it will be expunged.
und	If you change your mind about deleting a message, use the Undelete Command.
exp	The Expunge Command permanently removes deleted messages. Mercury will ask for confirmation before expunging messages.
k	The Kill command is the same as a Delete Command followed by a Next Command.

To issue a command that will act on other messages besides the current one, you type additional information after the command. This additional information is known as a command argument. For now, the only argument you need is the number of a message. Mercury assigns a number to a message according to the order in which Mercury received it.

h all	The Display All Headers Command allows you to find the number of a message. Mercury will print out short descriptions of the messages you have on file. These short descriptions are known as headers. The first field of the short header is the message number. Try this command before reading on.

Once you know the number of a message, you can give message numbers as command arguments.

Examples:

d 3	deletes the third message in your file.
t 2	displays the second message in your file.

A final note on message numbers

After a message has been expunged from your message file, the numbers of the remaining messages change. So, for instance, if you expunge the third message in your file, message 4 will become message 3, message 5 will become message 4, etc.

4.4 Reading Mail and Searching through Message Files

4.4.1 Other Commands for Reading Mail

When you invoke Mercury, it checks your mailbox for new messages and displays their headers. If you have received new mail, use the Type Command, *t* to display it. Use the Type Command followed by the number of a message to display any other message. If you have forgotten the number of the message, type *h all* to get a list of the short headers of your messages. The first field of each header is the number of the message.

4.4.2 Message Headers and Attributes

Each message is preceded by a header which is a summary of its contents. A header has six fields:

1. The NUMBER of the message, assigned according to when it was received.
2. "Flags" which tell you about ATTRIBUTES of the message. The flags are:

```
+   the message is NEW.
?   "       "     has not been ANSWERED.
-   "       "    .has not been EXAMINED.
*   "       "     has been DELETED.
B   You kept a BLIND message - you saved a copy of the
    message in your message file.
D   You kept a DRAFT of the message.
```

3. The DATE the message was sent.
4. Who the message was FROM.
5. The SUBJECT of the message.
6. The NUMBER OF CHARACTERS in the message.

4.4.3 Basic Search Commands

The commands given in the previous pages are sufficient for getting started with Mercury. However, when you become familiar with the system, and when you start accumulating a lot of mail, the information in the following pages will be of help. In the previous section the Display All Headers Command, *h all*, was introduced so that you could find the number of a message and use it as a command argument. All message attributes can be used in the same way as message numbers. For example, the command *t deleted* causes all the deleted messages in the file to be displayed on the screen. Remember to hit RETURN after typing a command.

There are other commands for examining messages. If you do not give an argument after a command, Mercury will act on the current message. The Current Command, *cu*, prints a short description of the current message. It also displays the name and number of messages in the file. The Goto Command, *g* followed by the name of an attribute, makes the last message containing the specified attribute the current message. Use the Type Command, *ty*, to display messages you make current with Goto. The Headers Command, *h* followed by an attribute, prints short headers of the messages you specify. The Wholeheaders Command, *w* followed by an attribute, prints the long version of message headers on the screen.

Message Numbers

As already explained, the number of a message is one of its attributes. You can make use of message numbers in a variety of ways. For example, *d 2,4,6* will delete messages 2, 4, and 6. If you specify two numbers separated by a colon, you receive the message corresponding to the number on the colon's left-hand side and all the messages up to and including the message corresponding to the number on the colon's right-hand side. For example, *2:8* displays messages 2, 3, 4, 5, 6, 7, 8. Typing an exclamation point between two numbers gives you the first-number message and as many messages following it as the value of the second number. The exclamation point is different from the colon. The number on the exclamation point's right-hand side is a numerical value, not a message number. For example, typing *2!3* will cause messages 2, 3, and 4 to appear on the screen.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Explained the purpose of the chapter, especially since it did not appear in the original.
- Eliminated explanations that the reader said were most confusing and least helpful. Moved other details that confused the reader to the more relevant parts of the text.
- Deleted distracting comments, e.g., attempts at humor.
- Deleted the Mercury syntax section because it came too early in the text.
- Assumed less experience and background knowledge on the part of the reader.
- Changed the format for commands: from abstract explanations to specific, verbatim instructions for invoking commands.
- Reorganized the chapter so that the reader received the most important information first; e.g., how to invoke and leave the system and how to get help.

STRATEGIES FOR REVISORS:

- Estimating the level of sophistication users will bring to the text is a critical concern for writers of documentation. Even when writers know specific characteristics of the intended audience, it is still quite difficult to write to those particular needs. But when writers are forced to write to mixed audiences, that is, an audience comprised of both novices and expert users, the task of choosing the most suitable organization, format, and language becomes extremely complex. Writers are often constrained by time, budget, and space demands--constraints that sometimes override issues such as providing the best manual for the target audience. The example text of this unit was written by a programmer who felt the text was simple and to the point. The revised text of this unit was written by a writer who was told "to adapt it for a mixed audience of programmers, secretaries, and professors." As you can see, although the text is greatly improved, it still may not allow flexible access for all potential users. Perhaps the best advice in such situations, is to start with the simplest way to operate the system and then to address more complex commands--since the most basic aspects of a system are usually the more important ones and ones the reader will use most often. Notice how the writer placed "emergency" information, such as how to exit the program and how to "get help", up front in the text.

- Users need to know when information in a text is beyond or beneath their level of sophistication. Generally, it is not advisable to shift modes of presentation within the body of the same text. As explained above, the writer of this text had no choice, and we provide this example to demonstrate how difficult writing to a mixed audience can be. Although the revisor provided caveats such as, "There are other ways to create messages, but if you are using Mercury for the first time it is best to use the Mail command. If you know how to give command arguments using message attributes and keywords, skip the rest of this section and go on to section 4.3.2.," the message functions as an interrupter more than practical advice.

- A summary of basic commands helps users to access information quickly. A clear way to present command summaries is to block information into groups, beginning with, for example, start up and exit commands and ending with those that require expert knowledge. You will notice the revision does not include the "Mercury Syntax" information; the revisor felt the basic features of the syntax commands were unnecessary in the first seven pages of a text aimed at a mixed audience.

UNIT 7

Context: The example text of Unit 7, **"The Mail System"** is part of an early draft of a tutorial on a mail system, "MS," operating on the TOPS-20 system. It was written for inclusion in the second volume of the Carnegie-Mellon TOPS-20 Manual. Readers are assumed to have a general experience with other aspects of the TOPS-20 system such as using EMACS, but no prior experience with "MS".

EXAMPLE TEXT:

The Mail System

MS is the name of the electronic mail system operated at CMU and run on the TOPS-20 computers. Through MS you can send or receive mail from anyone who has an account on any CMU TOPS-20 computer.

This document will serve as your introduction the the MS system and its most common uses. You can get along well in MS using only the features described here. As explained in detail in the section "Learning About TOPS-20," there are three ways the computer itself can give you help:

- If you type a ? after any MS prompt (such as MS> or MS read>>), the computer will respond with a list of valid responses for you to choose among.
- If you hit the `Escape` key (labeled **ESC** or `Altmode` on some terminals) after you have typed a command that requires an argument (such as MS>copy), the computer will type "guide words" which indicate what to type next (example: `MS>copy Into file.`)
- If you would like to receive basic information about MS, at the MS> prompt type *help novice*

Helpful Hint: A prompt is a character or a message on the screen that indicates where you are. The @ prompt indicates that you are at the EXEC (main) level, that is, you are not within any program. MS is a program, and like other programs, it contains levels. You enter through the top level (you type `ms` and receive the prompt `MS>`). At the top level you have a choice of many options; you type a command to indicate which option you want to take (for example, at the MS> prompt, you could type `send`). Then you will be at the second level (in the "send" example, the prompt will be `MS send>>`). There are many options available at this level also.

The chart in Appendix A lists all the things you can do in MS and tells you haw to get help from the system to learn about the options not covered in this manual.

How to enter MS

Action. At the EXEC level (@prompt), type ms

Response. If you have never been in MS before, the MS> prompt appears. (It may take a few seconds. Otherwise, a summary such as the following appears:

```
N 15 28-Jan CJ19 at TOPSC Thursday meeting
 (140 chars)  Last read:  28-Jan-82 13:59
 15 messages (14 old), 7 pages.
 Messages 1, 8 flagged.
```

In the above example the user has one new message, from CJ19. The user last read mail on January 28 at 5:59 p.m. MS now contains 15 messages (14 old plus the 1 new) and occupies 7 computer pages. The user has flagged Message Nos. 1 and 8 for easy retrieval (explained later).

After the message the MS> prompt appears.

Outgoing Mail

How to send messages

Overview

Briefly, here are the steps to follow. Each is explained in detail below.

1. Type ms
2. Type send
3. Type the user-id of the recipient(s).
4. Type the user-id of the person(s) to receive copies (optional).
5. Type a subject line (optional but recommended).
6. Type the message.
7. Choose one of the options given on the screen for handling the message: send the message (control z); change a header line (ESCAPE); edit the body

of the message (control e); redisplay
the message (control k); insert
another file into the body (control b);
abort the message (control n).

A note about addressing mail.

Before you can send mail, you should know the 4-character user-id of the person to whom you want to send a message and the TOPS system on which the person has an account. The section on the WHERE command tells you how to find this information.

Once you have entered MS, you have two recourses:

1) You can address mail to people using their names, but this is tricky--you must type their names exactly the way the computer has the names recorded and you must know the system on which they have an account.

2) You can use the translate command to find user-id's for only those people who have accounts on the system on which you are logged in (and again, you have to type their names as the computer knows them). To use this command, at the MS> prompt, type *translate* followed by the person's name and hit the RETURN key.

Once you have this information proceed as described below.

1. Action. Enter MS as explained previously.

2. Action. At the MS> prompt type *send* and hit the RETURN key.

   Response. The prompt says To:

3. Action. Address the message in one of the following ways:

   - If the person to whom you're sending mail is on the same TOPS system as you are, just type the user-id (example: jh12) and hit the RETURN key.
   - If the person is on another system, type their user-id followed by an @ sign and the system name (example: jh12@topsc) and hit the RETURN key.
   - If you want to address the same message to more than one person, type the user-id's separated by commas (example: jh12,cj19@topsd,lb4v) and hit the RETURN key.

   Response. If there is no problem,
   the prompt says *cc:*

   If the person is not on your system and
   you did not specify another system, or
   if there is any other problem, the response

will say `?no such user as "---"`

4. <u>Action.</u> If you want to send a copy to anyone, type the user-id (using one of the formats explained above for addressing a message) and hit the RETURN key. (You can send a copy to yourself if you'd like, by including yourself in the cc:line. For other ways to keep a copy, see the subsection "How to save copies of outgoing messages".)

   <u>Response.</u> The prompt says `Subject:`

5. <u>Action.</u> Type a line that tells in a few words what the message is about, and hit the RETURN key. Remember, though, that a subject line is very useful to the recipient.

   <u>Response.</u> The prompt will say:

```
ESC to enter Send level, ctrl-Z
 to send, ctrl-K to redisplay, ctrl-B
 to insert file, ctrl-E to enter editor,
 ctrl-N to abort:
```

Helpful Hint: At this point you can send a blind copy to someone by hitting the ESCAPE key and then at the `ms send>>` prompt typing *bcc* and hitting the RETURN key. The prompt will say `bcc:` Type the user-id of the person or persons to receive a bcc, hit the RETURN key, and then at the `ms send>>` prompt, type *text* to return to the body of the message.

6. <u>Action.</u> Type your message, hitting the RETURN key at the end of each screen line.

   <u>Response.</u> The message appears on the screen as it is typed.

If you make a typing mistake you can erase one character at a time with the DELETE key, one word at a time with control w, or one line at a time with control u. You can move backward through the message, even from line to line, by repeating any of these three commands. However, you <u>cannot</u> move from the body of the message back into the header lines, or vice versa.

USER PROTOCOL OF EXAMPLE:

MS is the name of the electronic mail system operated at CMU and run on the TOPS-20 computers. Through MS you can send or receive mail from anyone who has an account on any CMU TOPS-20 computer. This document will serve as your introduction the the MS system and its most common uses. You can get along well in MS using only the features described here. As explained in detail in the section "Learning About TOPS-20," there are three ways the computer itself can give you help: If you type a ? after any MS prompt (such as MS> or MS read>>), the computer will respond with a list of valid responses for you to choose among. If you hit the Escape key (labeled ESC or Altmode on some terminals) after you have typed a command that requires an argument (such as MS>copy), the computer will type "guide words" which indicate what to type next (example: MS>copy Into file.) If you would like to receive basic information about MS, at the MS> prompt type *help novice. No, I just want to read what's here.*

A prompt is a character or a message on the screen that indicates where you are. The @ prompt indicates that you are at the EXEC (main) level, that is, you are not within any program. MS is a program, and like other programs, it contains levels. You enter through the top level (you type ms and receive the prompt MS>). At the top level you have a choice of many options; you type a command to indicate which option you want to take (for example, at the MS> prompt, you could type send). Then you will be at the second level (in the "send" example, the prompt will be MS send>>). There are many options available at this level also. The chart in Appendix A lists all the things you can do in MS and tells you how to get help from the system to learn about the options not covered in this manual. *O.K., so now I've got a prompt and I type in MS, and I wonder if I have to hit a return? It doesn't say.*

O.k., so I have a whole series of messages. 311 characters, wonder what that means? It says "user has a message. User last read mail on 1-29." So I guess that means I read my mail on 1-29 six times, the last at 8:21? But there's no a.m. or p.m. here. The example says 5:59 p.m. *But on the screen it just says 8:21...I guess it should say a.m. or p.m., something must be wrong with my mail file. Well, I'll just skip this and go on to the overview.*

Outgoing Mail. How to send messages. Overview. Briefly, here are the steps to follow. Each is explained in detail below. *I can type MS and what do I get? Oh, these are single constructions. I thought they were options. First I type MS, then I type "send."*

I wonder if I need a space in between? I wonder if I hit Return? Type ms. Type send. Type the user-id of the recipients. Type the user-id of the persons to receive copies (optional). Type a subject line. Type the message. Choose one of the options given on the screen for handling the message. *I hit control Z, o.k. Now I change a header line. What's a header line? Is that a subject line or is it different? Well, go on...* Edit the body of the message control e; redisplay the message control k. *Why would I want to redisplay it, after I edit it?* Insert another file into the body (control b). *I just don't know what that means.* Abort the message control n. *That's just a complete wipe out then? O.k.*

Before you can send mail, you should know the 4-character user I.D.to whom you want to send a message and the TOPS system on which the person has an account. The section on the WHERE command tells you how to find this information.

Once you have entered MS, you have two recourses. You can address mail to people using their names, but this is tricky--you must type their names exactly the way the computer has the names recorded and you must know the system on which they have an account. You can use the translate command to find user-id's for only those people who have accounts on the system on which you are logged in and again, you have to type their names as the computer knows them. To use this command, at the MS prompt, type translate followed by the person's name and hit the return key.

Oops, I'm not taking all this in at all. Let's see, once I've entered MS I can address mail to people using their names. This is tricky. I still don't understand what it means to translate someone's name. Do I just type the last name or the full name? Well I'll try Mr. Adams--t-r-a-n-s-l-a-t-e A-d-a-m-s. Oh no, now it's given me 7 different Adams'. Well I guess I do need the full name.

Now there's no heading for what follows. Once you have this information proceed as described below. Enter MS as explained previously *O.K., so I type Ms and hit the return key. Oops, it says MS command error. Does not match switch or keyword. What did I do? Oh, I guess I'm already in MS. They should have told me that.* At the MS> prompt type send and hit the return key. *O.k., so they're asking me to whom I will send the message.* The prompt says To:>. If the person to whom you're sending mail is on the same TOPS system as you are, just type the user-id (example: jh12) and hit the Return key. If the person is on another system, type their user-id followed by an @ sign and the system name (example: jh12@topsc) and hit the return key.

If you want to address the same message to more than one person, type the user-id's separated by commas (example: jh12,cj19@topsd.lb4v) and hit the return key. If there is no problem, the prompt says cc:. *What does that mean, if there's no problem. Why should there be a problem if these are directions on how to do it? Now what I'm not picking up is if you look at the third bullet...oh, oh. I see. I thought that I was addressing several people, but I'm not. I'm just addressing jh12 and cj19 on topsd. Now what does LB4V mean? Does this mean I simply want to send copies to these others? I guess there is a difference between addressing three people directly and sending copies to three people.*

What if I don't want to send any copies? I guess I'll hit return and see what happens? O.k. that gives me subject. Type a line that tells in a few words what the message is about, and hit the Return key. Remember, though, that a subject line is very useful to the recipient. *Well, I've already done that and I' not doing it again.* The prompt will say: ESC to enter Send level, ctrl-Z to send, ctrl-K to redisplay, ctrl-B to insert file, ctrl-E to enter editor, ctrl-N to abort. *o.k., I still don't know what insert file means. What about the sending level? Does that mean just the subject line? I've entered my subject line and the prompt tells me to hit ESC to enter "send" level. I wonder if this was explained earlier and I missed it? I don't think so but I'll look. No...don't see it. I don't know whether to hit ESC or not.*

Helpful Hint: At this point you can send a blind copy to someone by hitting the ESCAPE key and then at the ms send>> prompt typing bcc and hitting the return key. The prompt will say bcc: Type the user-id of the person or persons to receive a bcc, hit the return key, and then at the ms send>> prompt, type text to return to the body of the message. *I don't want to send a blind copy to anyone. Oh all right, hit escape. Oh no, command error. Does not match switch or keyword. Oh well. Oops, I hit it again, my message is in and I didn't get to edit it. I know I made a typo and I forgot to do the following.* Type your message, hitting the return key at the end of each screen line.

If you make a typing mistake you can erase one character at a time with the delete key, one word at a time with control w, or one line at a time with control u. You can move backward through the message, even from line to line, by repeating any of these three commands. However, you cannot move from the body of the message back into the header lines, or vice versa. *Shit, I can't do any of this because my mail is already processed. I think. I don't understand what it means to move the body of the message into the header. It says processing mail. Local mail delivered. I wonder if once you've sent it if there's any way of taking it out? Or is it just like dropping it in a mailbox.? You know, I don't really remember anything I've just done. Now the question is if I had gone through just one group of commands, how much more would I have* *learned? I guess the real question is, should this mail section be written as a tutorial? If you aren't going to remember it then why bother with all the steps? I don't know if I really need to know all these options and hints at this point. Maybe it should be part tutorial and part something else. I'm so confused with new information that I can't even remember how to log out. Where are my instructions? Oh, I guess I've got to get up to the executive level. Somewhere there are instructions. I remember what the page looks like but I'll be damned if I can find it. Part of the problem with this dumb manual is that you can't find your way around it. The whole business makes you feel stupid.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- Do I hit return after the @prompt?
- What do characters mean?
- Is there something wrong with my mail file if there is no "a.m." or "p.m." after the time?
- What is an option in the mail system?
- What is a header line? Is it the same as a subject line?
- Why would I want to redisplay a letter after it has been edited?
- What does it mean to translate someone's name?
- Why does the text say "if there is no problem" the prompt will say "cc"?
- What is the difference between sending copies to three people and addressing them directly?
- What if I don't want to send copies?
- What does insert file mean?
- What is the sending level?
- What does it mean to move the body of the message to the header?
- Once you've sent mail, is there any way of taking it out?
- Should this section be written as a tutorial? If you aren't going to remember it, then why bother with all the steps?
- Do I really need to know these options?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

THE MAIL SYSTEM

MS is the name of the program that is used to handle electronic mail on the CMU TOPS-20 systems. Using MS you can send or receive messages from other users.

When you receive a message from another user, it is added to a file called Mail.txt in your directory. Using MS you can read messages kept in the Mail.txt file and perform functions such as sending mail to someone else (that is, putting a message into their Mail.txt file, replying to the sender of a message, forwarding a message that you received to someone else, or deleting messages that you don't need anymore.

Messages stay in you Mail.txt until you tell MS to delete them. When you delete a message, MS removes it from the Mail.txt file. If there are no messages left in the Mail.txt file, MS removes the file from your directory. When you receive a message, Mail.txt is created again. It is unwise to delete the Mail.txt file as a whole because you might throw away messages that you haven't seen or replied to; do you deleting with MS.

MS, like other programs, contains levels of commands. You enter by typing `ms` at the Exec. You then receive the MS> prompt, which indicates that you are at the top level of MS. At the MS> prompt, you have a choice of many options. If you type `send` or `read` you will go the second level (in the "send" example, the prompt will be MS send>>). There are many options available at the level also.

This document contains a tutorial to introduce you to the most basic functions of MS and a reference section which explains basic MS in more detail and covers some of the more advanced features.

Tutorial

Goals: In this tutorial you will learn how to enter MS, the basic procedures to send a message and to read an incoming message, and how to leave MS. When you go through the tutorial the first time, ignore the references to other pages. After you have completed the tutorial at least once and feel comfortable with MS, go to the pages referenced to learn more about the features of MS.

In this tutorial you will be sending a message to a friend, with a copy to yourself. If you know that one of your friends has a computer account and you know what system the account is on (A, B, C, etc.), proceed to enter MS. If not, use the WHERE program described in Volume 1, Tutorial B - **Learning About TOPS-20**, to locate a friend to whom you wish to send a message.

TO ENTER MS:

Action 1. To enter MS, at the Exec @ prompt type ms and hit RETURN.

Response. The MS> prompt appears. (It may take a few seconds.)

TO SEND A MESSAGE:

Action 2. At the MS> prompt type send and hit RETURN.

Response. The prompt says To:

Action 3. Address the message to one of your friends who has a computer account, as explained below:

If you are logged into the same computer on which your friend has an account, type your friend's last name and hit the RETURN key. You do not need to specify the computer when the user has an account on the same computer that you are logged into.

If you are logged into some other computer, type your friend's last name followed by an @sign and the computer (example: smith@topsd) and hit the RETURN key. (Do not put spaces before or after @.)

Response. The prompt says cc:

- For more information on addressing messages, see page G-10.

Action 4. Send a copy to yourself by typing in your user-id and hitting the RETURN key. (This will provide you with a message to use below when you learn how to read incoming messages.)

The cc line is optional. In the future if you don't want to send a copy to anyone, hit the RETURN key.

Response. The prompt says `subject`

- For more information on the cc line, see page G-11.

- For more information on keeping copies of your messages, see page G-17.

Action 5. Make up a subject line (a few words that tell what the message is about), type it, and hit the RETURN key.

The subject line is optional; you can skip it by hitting the RETURN key. However, a subject line is very helpful to the recipient, as explained on page G-11.

Response. The message prompt appears (the word `message`, followed by all the options available). Ignore the options for now.

- The options are explained on pages G-13 through G-15.

Action 6. Type a brief message. Hit the RETURN key at the end of each screen line. (If you don't hit the RETURN, MS may start a new line in the middle of a word. The recipient will receive the message with these awkward breaks.)

Response. The message appears on the screen as it is typed.

- For more information on typing and correcting the body, see page G-13.

Reference Section
Processing Outgoing Mail

Overview: To send messages, you enter MS and issue the "send" command. Like internal memos, each message contains To, cc, and Subject lines. These are called the header lines, and you will be prompted for each. The date and the sender will be supplied by the system.

After you complete the header lines, you will be prompted for the body of the message. At that point you can ask for a bcc (blind carbon copy) line if you wish. Then type the body. You can correct typing mistakes as you type or make all corrections at once when you're done by going to the editing level. If you need to, you can change a header line, redisplay the message, insert another file into the body of the message, or abort the message altogether. Each of these options is explained in detail in this subsection.

After you send the message, you will receive confirmation from the system that the message has been sent.

How to type the header lines (To, cc, Subject)

The header lines of a message are to To, cc, and Subject lines. You will be prompted for each. Details about each line are given after the following information which applies to all three lines.

If you make a mistake while typing header lines, use the following procedures to correct it.

Action. If you realize that you have made a mistake while you are still on the line, correct it using the DELETE key (for one character), CTRL W (for one word), or CTRL u (for one line).

If you have left the line, either

1) do a CTRL n to abort the message and start over or

2) go ahead and type all of the header lines. Then at the `Message` prompt or at any point in the body of the message, hit the ESC key.

(You will return to the body after you have corrected the header line.)

Response. The prompt says

```
MS send>>
```

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Deleted complex information from the introduction.
- Explained how a file called Mail.txt is added to the user's directory when someone sends a message.
- Deleted the "helpful hint" from the introduction.
- Explained the document's goals: a basic tutorial and a reference section.
- Deleted the section "Outgoing Mail--an Overview.
- Moved optional and more complex information to the reference section.
- Provided cross-references on where to obtain detailed information.
- Consolidated the important features needed to get started in the mail program.

STRATEGIES FOR REVISORS:

- Although tutorials (i.e. action-response formats) best serve the needs of users across many situations, they sometimes overburden the reader's memory, especially when they provide too much optional information. Writer's need to be sensitive to the thin line between "providing necessary information" and "bombarding the reader with too much detail". When a text's purpose is to get a user started in a program, make discriminations between information that must be included and that which can be referenced or placed in a more advanced section. Provide the key commands and tell the user where options can be located.

- When providing information that has been condensed from a large and detailed text, it is important to clarify the text's scope through a purpose or goal statement. Tell the user how much of the text he/she needs to cover in order to gain a basic understanding of its features. Notice how this unit's revision asks the reader to use the tutorial the first time by going through steps 1-6 and by ignoring references. This strategy makes the task considerably easier for novices. It also it alerts experts to the text's level of sophistication and where to find advanced information.

- When providing a reference section, briefly summarize the main points to be covered in an overview. Assume the reader has read or has prior knowledge of the subject matter the text builds on. Organize the section so that it follows the preview. If the material will be read by both novice and expert users, you may still want to include a tutorial. To illustrate this point, we included only the first page of the "mail" program's reference section in this unit. Notice the topics covered are those that most confused the reader during the user protocol.

UNIT 8

Context: The example text of Unit 8, "**Introduction to Oil**" describes an editor, "Oil," which is part of an advanced programming environment being developed at Carnegie-Mellon. The text is taken from an internal document written by the program developer. At the time it was written, it was the only document describing "Oil". Therefore, one might expect that the intended audience is a general one. The revision is written as a tutorial for a mixed audience--one that was described to the revisor as consisting of new graduate students in computer science and secretaries in the department of engineering and computer science.

EXAMPLE TEXT:

Oil: The Introduction

OIL is yet another text editor. It incorporates features of many previous editors, but is not quite like any of them. The goals of the OIL editor are:

- To fit into the Spice environment. In particular to access the screen through Canvas [Ball 81] and to take advantage of the large virtual storage provided by Accent [Rashid 80].
- To be available quickly, at least in skeletal form. In order for program development under Accent to be possible there must be *some* editor available. The first version of OIL is not meant to be the last word in editors, but was designed to be quickly implementable.,
- To be smoothly expandable. The first version of OIL is not the last. Features which are not yet implemented, but which are anticipated include programmability as in Emacs [Gosling 81], interface to MultiScript [Multi 81] and (possibly) multiple character fonts.

This document concentrates on describing the editor as it is now, but there is a section near the end on future plans.

The Basics

OIL is a modeless editor. This means that pressing the same key will always cause the same effect. This statement has to be interpreted somewhat liberally. The ctrl-x key serves as an escape, and changes the meaning of the next key pressed. The ctrl-x and the following key must be interpreted as a single compound keystroke.

The focus of attention when using OIL is the "current position." The current position is always between two characters, and is marked by a short vertical bar called the "pencil." Most commands act on the characters immediately preceding or following the pencil.

There is a cursor which follows the mouse. This cursor arrow can be used to point to a position in the text so as to move the pencil to that position, but the position of the cursor arrow does not directly affect the text editing.

The OIL editor does not recognize the existence of trailing blanks. No file written by the OIL editor will contain blanks at the end of a line, or totally blank lines at the end of the file. The cursor arrow roams freely over the entire screen, but it is not possible to move the pencil to a position outside of the visible text. It is, of course, possible to type blanks at the end of a line, but these blanks are ephemeral. Unless a visible character is typed after them, they will disappear when the pencil leaves the line.

The Display

The OIL editor provides two kinds of windows, prompt and text. The single prompt window displays information on the current state of the editor, including error messages, questions, and the contents of the kill buffer.

One or more text windows are also displayed. Each window is associated with a file of text. A text window is subdivided into three areas: text, a horizontal "status bar" above the text, and a vertical "scroll bar" to the left of the text. The status bar contains the name of the window (generally the same as the name of the file being edited), and a "thumb bar". The cursor arrow functions differently when it is in the scroll bar or thumb bar from when it is in the text region. The cursor arrow may move freely from one text window to another.

Use of the Mouse

The Summagraphics bit-pad that is used with the Perq is intended to be used only as an absolute tracking device--it can only report the current position of the puck. For most purposes relative tracking is more convenient, it has therefore been implemented in software (within Canvas). This permits the puck to be used in the same way as the mouse on the Alto. The mouse can be picked up and moved to a more convenient position without changing the position of the cursor on the screen.

The mouse is a convenient, though not essential, source of commands. The mouse can be used to scroll through the text and to change the current position. However the same thing can be accomplished using the keyboard.

File Conventions

OIL can be invoked by typing "oil<filename1><filename2>..." to the shell. Each of these filenames will be used in turn. Just before each file is read the editor displays the file name and allows it to be edited. Filenames are edited using exactly the same commands as for text, except that commands that would cause motion between lines are not allowed. The extensions ".pas", ".mss", ".cmd", and "micro" will be concatenated to the given name in an attempt to find the file. If there is no file with that name an empty file will be created.

The Kill Ring

OIL does not provide an explicit undo command. However it is possible to undo any command except deletion of windows and text replacement. For most commands there is an obvious complementary undo command. Yank-buffer is the complementary undo for delete commands of all kinds. Every time text is deleted it is saved in a kill buffer. If the cursor has not been moved since the last delete command then the deleted text is appended to the current kill buffer. If the cursor has been moved, the deleted text replaces the contents of the kill buffer. However the text that was in the kill buffer is not lost, it is simply pushed around the kill ring.

The kill ring consists of four buffers, only one of which is displayed in the prompt window. When text is deleted at a new position, it overwrites the oldest buffer in the kill ring. In other words the last four groups of deletions are saved, a group being any number of delete commands which are done without moving the cursor. The yank-buffer command inserts the displayed kill buffer into the file at the current position. There is no way to directly access the other buffers in the kill ring. To get at these other buffers one uses the pop-kill-ring command. This command moves the top (displayed) kill buffer into the last position of the kill ring. The second kill buffer then becomes the first, and is displayed in the prompt window. There are four buffers in the kill ring, and so four pop-kill-ring commands will return the editor to its original state.

Key Bindings

Since OIL uses Canvas to get commands from the keyboard, the Canvas key translation facilities determine the binding of keys to commands. The commands are described below in terms of the Emacs style bindings, but it should be kept in mind that the specific key used to invoke each command can be altered. The binding of keys to commands is determined by the file "oil.keytran". This file can be changed to suit the habits of a particular user by using the key translation editor. There is no need to recompile the editor to do this.

The Commands

We now come to a detailed list of the available commands. Remember that all ordinary alphanumeric keys simply cause the appropriate character to be inserted in the file. To invoke an editor command hold down either the "ctrl" key or the yellow mouse button while pressing a key. This will not be repeated for every command.

The Mouse and Buttons

When the cursor arrow is in the text area pressing the yellow mouse button will move the pencil to the position pointed to. If the button is held down then any keys typed with the other hand will be interpreted as editor commands. If the button is released then the keys will simply be inserted into the file at the new position of the pencil.

The thumb bar above the text region allows rapid access to distant parts of the file. The file is linearly mapped onto the horizontal bar with the part of the file visible in the text window represented in black. To go to a new location in the file simply point the cursor arrow at the appropriate part of the thumb bar and press the yellow mouse button.

The scroll bar allows sequential motion through the file. Three kinds of scrolling are provided. To invoke scrolling first move the cursor arrow into the scroll bar. Pressing the yellow (top) button then moves the line next to the cursor to the top of the window. Pressing the blue (bottom) button moves the line at the top of the window to the position of the cursor.

Smooth scrolling can be invoked by pressing and holding the white (left) button. Once the white button is pressed scrolling continues until it is released. The magnitude and direction of scrolling is controlled by the position of the cursor with respect to its position when the button was originally pressed. Nothing happens until the cursor is moved. As the cursor is moved further from its original position scrolling speeds up. If the cursor is returned to its original position or the button is released scrolling stops.

USER PROTOCOL OF EXAMPLE:

The Introduction. OIL is yet another text editor. *I guess the writer is bored with text editors.* It incorporates features of many previous editors, but is not quite like any of them. *What kind of editor is Oil similar to?* The goals of the OIL editor are to fit into the Spice environment. *What is the Spice environment?* In particular to access the screen through Canvas. *Canvas?* and to take advantage of the large virtual storage provided by Accent Rashid 80. *What does it mean to take advantage of the virtual large storage space?* To be available quickly, at least in skeletal form. In order for program development under Accent to be possible there must be *some* editor available. The first version of OIL is not meant to be the last word in editors, but was designed to be quickly implementable. To be smoothly expandable. The first version of OIL is not the last. Features which are not yet implemented, but which are anticipated include programmability as in Emacs Gosling 81, interface to MultiScript and (possibly) multiple character fonts. This document concentrates on describing the editor as it is now, but there is a section near the end on future plans. *Why tell me about future plans and anticipated programmability and interfaces? Do I need these to invoke Oil?*

The Basics. OIL is a modeless editor. *What is a modeless editor?* This means that pressing the same key will always cause the same effect. This statement has to be interpreted somewhat liberally. *How do you "somewhat liberally interpret" a statement?* The ctrl-x key serves as an escape, and changes the meaning of the next key pressed. The ctrl-x and the following key must be interpreted as a single compound keystroke. *O.K., that sounds familiar to what I'm used to.*

The focus of attention when using OIL is the "current position." *What is the focus of attention?* The current position is always between two characters, and is marked by a short vertical bar called the "pencil." *Are the pencil and the cursor the same?* Most commands act on the characters immediately preceding or following the pencil. *I wonder if that means that some commands don't.*

There is a cursor which follows the mouse. *How does the cursor "follow" the mouse?* This cursor arrow can be used to point to a position in the text so as to move the pencil to that position, but the position of the cursor arrow does not directly affect the text editing.

The OIL editor does not recognize the existence of trailing blanks. *What is a trailing blank?* No file written by the OIL editor will contain blanks at the end of a line, or totally blank lines at the end of the file. The cursor arrow roams freely over the entire screen. *Roams freely? I thought it only moved if I did something.* But it is not possible to move the pencil to a position outside of the visible text. It is, of course, possible to type blanks at the end of a line, but these blanks are ephemeral. *How are blanks at the end of a line ephemeral?* Unless a visible character is typed after them, they will disappear when the pencil leaves the line.

The Display

The OIL editor provides two kinds of windows, prompt and text. The single prompt window displays information on the current state of the editor, including error messages, questions, and the contents of the kill buffer. *What is meant by the "current state" of the editor?*

One or more text windows are also displayed. Each window is associated with a file of text. A text window is subdivided into three areas: text, a horizontal "status bar" above the text, and a vertical "scroll bar" to the left of the text. The status bar contains the name of the window (generally the same as the name of the file being edited), and a "thumb bar". The cursor arrow functions differently when it is in the scroll bar or thumb bar from when it is in the text region. *How are the bar and text region areas different?* The cursor arrow may move freely from one text window to another. *Roam freely? How are windows associated with files?*

Use of the Mouse. The Summagraphics bit-pad that is used with the Perq is intended to be used only as an absolute tracking device--it can only report the current position of the puck. *I guess the bit pad is this little table.* For most purposes relative tracking is more convenient, it has therefore been implemented in software (within Canvas). *What is the difference between an absolute tracking device and a relative tracking device?* This permits the puck to be used in the same way as the mouse on the Alto. *What is the Alto? Sounds like a terminal that I've never used.* The mouse can be picked up and moved to a more convenient position without changing the position of the cursor on the screen.

The mouse is a convenient, though not essential, source of commands. *If the mouse is not essential, then why use it?* The mouse can be used to scroll through the text and to change the current position. However the same thing can be accomplished using the keyboard.

File Conventions. OIL can be invoked by typing "oil<filename1><filename2>..." to the shell. *What is a shell?* Each of these filenames will be used in turn. Just before each file is read the editor displays the file name and allows it to be edited. Filenames are edited using exactly the same commands as for text, except that commands that would cause motion between lines are not allowed. The extensions ".pas", ".mss", ".cmd", and "micro" will be concatenated to the given name in an attempt to find the file. If there is no file with that name an empty file will be created. *O.k., sounds reasonable but I still can't do anything yet. I don't know if I'll be able to remember all this without executing something. Oh well, I just keep going and hope I don't forget.*

The Kill Ring. OIL does not provide an explicit undo command. *Are undo and undelete the same?* However it is possible to undo any command except deletion of windows and text replacement. For most commands there is an obvious complementary undo command. Yank-buffer is the complementary undo for delete commands of all kinds. Every time text is deleted it is saved in a kill buffer. If the cursor has not been moved since the last delete command then the deleted text is appended to the current kill buffer. If the cursor has been moved, the deleted text replaces the contents of the kill buffer. However the text that was in the kill buffer is not lost, it is simply pushed around the kill ring. *How is text in the kill ring "pushed around"?*

The kill ring consists of four buffers, only one of which is displayed in the prompt window. When text is deleted at a new position, it overwrites the oldest buffer in the kill ring. In other words the last four groups of deletions are saved, a group being any number of delete commands which are done without moving the cursor. The yank-buffer command inserts the displayed kill buffer into the file at the current position. There is no way to directly access the other buffers in the kill ring. *This is quite complex and I can't really picture it.* To get at these other buffers one uses the pop-kill-ring command. *How do you "pop" the kill ring?* This command moves the top (displayed) kill buffer into the last position of the kill ring. The second kill buffer then becomes the first, and is displayed in the prompt window. There are four buffers in the kill ring, and so four pop-kill-ring commands will return the editor to its original state. *I'm confused here.*

Key Bindings. *What is a key binding?* Since OIL uses Canvas to get commands from the keyboard, the Canvas key translation facilities determine the binding of keys to commands. The commands are described below in terms of the Emacs style bindings, but it should be kept in mind that the specific key used to invoke each command can be altered. The binding of keys to commands is determined by the file "oil.keytran". This file can be changed to suit the habits of a particular user by using the key translation editor. There is no need to recompile the editor to do this. *Why do I need to think about recompiling, even if I knew how to do it?*

The Commands. We now come to a detailed list of the available commands. Remember that all ordinary alphanumeric keys simply cause the appropriate character to be inserted in the file. To invoke an editor command hold down either the "ctrl" key or the yellow mouse button while pressing a key. This will not be repeated for every command. *Is the CTRL key or the mouse button more appropriate?*

The Mouse and Buttons. When the cursor arrow is in the text area pressing the yellow mouse button will move the pencil to the position pointed to. If the button is held down then any keys typed with the other hand will be interpreted as editor commands. If the button is released then the keys will simply be inserted into the file at the new position of the pencil.

The thumb bar above the text region allows rapid access to distant parts of the file. The file is linearly mapped onto the horizontal bar with the part of the file visible in the text window represented in black. *What does linearly mapped mean?* To go to a new location in the file simply point the cursor arrow at the appropriate part of the thumb bar and press the yellow mouse button.

The scroll bar allows sequential motion through the file. Three kinds of scrolling are provided. To invoke scrolling first move the cursor arrow into the scroll bar. Pressing the yellow (top) button then moves the line next to the cursor to the top of the window. Pressing the blue (bottom) button moves the line at the top of the window to the position of the cursor.

Smooth scrolling can be invoked by pressing and holding the white (left) button. *What is "smooth" scrolling?* Once the white button is pressed scrolling continues until it is released. The magnitude and direction of scrolling is controlled by the position of the cursor with respect to its position when the button was originally pressed. Nothing happens until the cursor is moved. As the cursor is moved further from its original position scrolling speeds up. If the cursor is returned to its original position or the button is released scrolling stops. *Well, that was certainly a mouthful. Although the system does not seem radically different from others I know I still have problems wading though this dense stuff. I feel unable to prioritize anything I've read--guess I can't see the trees from the woods.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- What kind of editor is Oil similar to?
- What is the Spice environment? What does it mean to take advantage of the virtual large storage space?
- Why tell me about future plans and anticipated programmability and interfaces? Do I need these to invoke Oil?
- What is a modeless editor?
- How do you "somewhat liberally interpret" a statement?
- What is the focus of attention?
- Are the pencil and the cursor the same?
- How does the cursor "follow" the mouse?
- What is a trailing blank? How are blanks at the end of a line ephemeral?
- What is meant by the "current state" of the editor?
- What is a kill buffer?
- How are windows associated with files?
- How are the bar and text region areas different?
- What is the difference between an absolute tracking device and a relative tracking device?
- Is a puck the same as a pencil?
- What is the Alto?
- If the mouse is not essential, then why use it?
- What is a shell?
- Are undo and undelete the same?
- How is text in the kill ring "pushed around"?
- How do you "pop" the kill ring?
- What is a key binding?
- Why do I need to think about recompiling, even if I knew how to do it?

- Is the CTRL key or the mouse button more appropriate?
- What does linearly mapped mean?
- What is "smooth" scrolling?
- How can I remember and prioritize what I've read?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

Chapter Two

Oil: The Spice Editor

2.1 Introduction

This chapter is about Oil, the editor that runs on Accent. Many of the Oil commands described in this chapter are similar to EMACS editing commands. Like Pepper (the editor that runs on POS), Oil allows you to position the cursor in two ways: with the pointing device or with the usual keyboard commands. One advantage of using Oil is that it allows you to take advantage of the large storage space provided by Accent.

2.2 The Basics

Read this section through before you use Oil. Then follow the instructions in Exercise 1 on the next page.

2.2.1 Invoking Oil

To create or edit a file

edit *<filename>* RETURN

You may abbreviate the edit command by typing *ed*. If you are creating a new file, you will need to hit RETURN a second time in order to use Oil.

After you type the filename and hit RETURN, Oil gives you a chance to edit the filename. You edit filenames using the same commands that you would use to edit text, except that you cannot use commands that move the cursor between lines of text.

If you do not specify an extension to your file name, Oil will add one for you (such as .pas, .pasmac, .mss, .cmd, or .micro).

2.2.2 Leaving Oil and Saving Your Work

To save a file and leave Oil

CTRL xf

If you are working in more than one window, each of the windows is deleted in turn and the editing session ends. As each window is deleted, Oil prompts you for the name of a file in which to save the changes you made. Your options at this point are to:

1. hit RETURN to save the file using the same name,

2. delete the filename, type in a new name, and hit RETURN to save the file under a new name, or

3. hit OOPS and RETURN to delete the filename and not save the file.

Exercise 1: Invoking Oil

Step One. If you haven't done so already, boot Accent and create a window that takes up about half of the gray area of the screen.

Step Two. *edit* *<filename>* RETURN

If you are creating a new file, hit RETURN again. If you are editing a file that already exists, you only need to hit RETURN once.

Read the description of the Oil screen display below.

2.2.3 The Oil Screen Display

Oil divides the window into two areas:

1. The small **prompt window** (at the top of the screen, below the ProcMan window) displays information on the current state of the editor, including error messages, questions, and the contents of the kill buffer.
2. The large **text window** takes up the rest of the screen. The text window is further subdivided into two areas:
   a. The gray horizontal bar at the top of the text window is the **thumb bar**. The left side of the thumb bar displays the name of the window (usually the same as the name of the file you are editing). The black bar inside the thumb bar indicates your current position in the file.
   b. The vertical bar to the left of the text window is the **scroll bar**, which allows you to move quickly through your file.

Like EMACS and Pepper, the default activity in Oil is inserting text. The cursor in the text window indicates your current position in the file. Whatever you type in appears to the left of the cursor.

Step Three. If this is the first time you are using Oil and you just created a new file, type in at least two screens full of text that you can use to practice the commands on the following pages.

2.3 Basic Oil Comands

Oil editing commands make use of the CTRL key. To invoke an Oil command, hold down the CTRL key while typing the CTRL character.

You may use the pointing device instead of CTRL to invoke an Oil command. To do so, hold down the middle button on the pointing device and complete the command by typing the CTRL character.

CTRL X is the meta command prefix. That is, doing a CTRL X is like hitting the ESC key in EMACS. CTRL X *character* means that you should hold down the CTRL key and then type X followed by whatever character completes the command.

2.3.1 Moving the Cursor

To move the cursor a short distance (that is, within the current screen) use the following commands. Letter case is important. For example, a lower case f moves the cursor forward one letter; a capital F moves the cursor forward one word.

Forward one character	CTRL f
	If the cursor is at the end of a line, moves the cursor to the beginning of the next line.
Forward one word	CTRL F
Back one character	CTRL b
	If the cursor is at the beginning of a line, moves the cursor to the end of the previous line.
Back one word	CTRL B
To the beginning of a line	CTRL a
	moves the cursor to the beginning of the current line.
To the end of a line	CTRL e
To the previous line	CTRL p
	moves the cursor to the same column in the previous line.
	Note: Oil remembers the column in which the cursor is located. If you move the cursor to the previous line, the cursor will stay in the same column, even if there is no text at that position. If the cursor is in the first line of the text window, the text will scroll down one line. If the cursor is at the top of the file, nothing happens.

To the next line	CTRL n	moves the cursor to the same column position in the next line. If the cursor is in the last line of the window, the text will scroll up one line. If there is no text at the new position, the next editing command will move the cursor to the end of the line. If the cursor is at the bottom of the file, nothing happens.
Back one page of text	CTRL V	the text scrolls back about 3/4 of a page; the cursor remains in the same row and column on the screen.
Forward one page of text	CTRL v	the text scrolls forward about 3/4 of a page; the cursor remains in the same row and column on the screen.
To the top of a window	CTRL ,	moves the cursor to the first character of the first line in the window.
To the bottom of a window	CTRL .	moves the cursor to the first character of the last line in the window.
To the top of a file	CTRL <	moves the cursor to the first character of the first line in the file.
To the bottom of a file	CRTL >	moves the cursor to the end of a file.

2.3.1.1 Using the Pointing Device to Move the Cursor

1. **To change the position of the cursor:**

   Use the pointing device to move the arrow to the area on the screen where you want the cursor to be, making sure that the arrow points to the exact spot you want. Then press and release the middle button on the pointing device. The cursor will move to the position you pointed to and anything you type after releasing the button will be inserted at the new position.

2. **To move the cursor to another file window:**

   Slide the pointing device around on the tablet until the arrow is in the new window. Then press and release the middle button on the pointing device.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Extracted the most essential features of the introduction and created analogies between editors with which the user was most familiar and Oil.
- Reformatted "The Basics" section so that it specifies what to read before using Oil.
- Gave the user the basic commands to try first. Separated invoking and leaving Oil and highlighted these commands through indenting the body of the text.
- Deleted unnecessary programming jargon.
- Created an exercise on invoking Oil to remind the reader what to remember about the system (e.g., booting Accent) before using Oil.
- Summarized the Oil screen display and divided information according to its function.
- Repositioned information on differences between the puck, the pencil, and the cursor to an early section of the manual.
- Designed a square box around the appropriate summary commands to contrast them with information on the display.
- Moved the section on the mouse, to a more appropriate introductory chapter on the Perq equipment.
- Created the following notation conventions:
    - UPPER CASE--indicates special function keys (such as CTRL, OOPS, INS, etc.) and means that you should hit the function key.
    - **Boldface**--is used in the text of the manual to highlight important terms and information.
    - *italics*--is used in the exercises to indicate what should be typed.
    - `typewriter font`--is used to represent what the system types on the monitor screen.
    - *<generic name>*--is used to indicate a generic name that should be replaced with a specific name.

STRATEGIES FOR REVISORS:

- People learn new information faster and easier when they can relate it to familiar or known information. If you are aware of the audience's prior knowledge of a subject, you can help the reader to see connections between old and new ideas through creating analogies. The revisor of this unit's example text, for instance, knew two important features about the audience: 1) they are familiar with using either Emacs or Pepper to edit their prose; and 2) they know little, if anything, about programming. The writer took these audience characteristics into account and revised on the basis of their needs. On the one hand, she created analogies between the known text editors and Oil. On the other, she deleted programming concepts--thus making the text more user friendly. Pointing out likenesses between programs helps experienced users to access relevant contexts and to make inferences they may otherwise miss.

- When providing complex procedural instructions, maintain consistent notation conventions to enable the user to see differences between information such what to type and what the system types on the monitor screen. Although the rationale for this strategy is obvious, manuals often confuse readers by including information about commands without telling the user if they are to simply to read what is presented or to do what is presented.

- When background or otherwise critical information needs to be included within the context of procedural instructions, contrastive formatting will help the reader to see differences quickly. . Notice how the revision of this unit visually separates the "getting started tutorial exercise from information about the display screen.

- Many computing situations do not require users to make use of expert knowledge about topics such as interface capacities, key bindings, and recompiling. When issues like these are not directly relevant to the tasks you expect the manual to teach, it is wise to either delete such information or to place it in another guide/section designed particularly for sophisticated users who may want to take advantage of such features.

UNIT 9

Context: The example text of Unit 9, "**ILS: Integrated Library System**" describes an on-line library catalog system. The example text is designed as a "getting started" tutorial in searching for library works using the LS/2000. Since on-line catalogs are a relatively recent innovation, the "ILS" tutorial is intended to help any user find library holdings such as books, journals, government documents, musical scores and records--holdings they would usually look up in a card catalog. The revised tutorial does not assume any prior experience with the LS/2000 nor with sophisticated library terms such as "LC subject headings".

EXAMPLE TEXT:

ILS: Integrated Library System

What is ILS

The Integrated Library System, known as ILS, provides access to a catalog of holdings of all branches of the University Libraries. The system is available to you from terminals placed in the libraries and currently contains about 8,000 test records.

ILS is designed to be used by new users who are not familiar with online catalogs. As a user you won't have to worry about destroying any part of the system or "breaking something". The system is designed to only DISPLAY any information you request of it.

ILS will PROMPT you as to the next step in your search and will tell you when you've done something wrong. It will also suggest ways to help you fix your mistakes. As you progress in searching ILS you will be able to shortcut these step by step directions and use the short commands on pages 10 and 11. If you need help you can type HELP followed by CARRIAGE RETURN; ILS will respond with instructions.

Getting Started and Sample Search

When you sit down at the TERMINAL it will either be displaying the first screen of ILS or part of the last person's search. You will need to start at the first screen. To do this, type /ES until the screen contains:

Choose the type of Search You Wish to Perform:

```
1 - BY AUTHOR
2 - BY TITLE
3 - CORPORATE/CONFERENCE
4 - OTHER
OR
ENTER A TERM
```

Suppose you wanted to find out if there are any books in ILS by Carl Sagan. You would type 1 (the number of a search by author) and <CR>. ILS will respond with a screen that has three spaces in which to type the author's name. Notice that the CURSOR is positioned at the beginning of the space where you are to enter information. Type the author's last name (Sagan) and a <CR>; the cursor moves to the next information block. Type the author's first name (Carl) and a <CR> (If you don't know the author's first name simply type <CR> here); the cursor moves to the beginning of the third information block. We don't know Carl Sagan's middle name, so just type <CR> ILS will respond with a screen that contains REFERENCE NUMBERS, the number of items that contain Carl Sagan as author, and the name of the author as it appears on the RECORD. At the bottom of the screen is:

```
CHOICE:R

Author:  Sagan, Carl

          REF             TITLES          AUTHOR

          R1                   5           SAGAN, CARL, 1934-

(END)

CHOICE:R
```

Notice that the cursor is now after the R at the bottom of the screen. If you had more than one choice you would enter the R number of the selection you wanted and type <CR>. Since there is only one choice here just press the <CR>. ILS will repond with a screen something like this:

```
Author:  Sagan, Carl, 1934-

      REF           TITLES                       AUTHOR

      R1              Planetary Atmospheres        Sagan, Carl, 1934-
      R2              UFO's-A Scientific Debate     Sagan, Carl, 1934-
      R3              The Cosmic Connection         Sagan, Carl, 1934-
      R4              Cosmos                          Sagan, Carl, 1934-
      R5              The Dragons of Eden           Sagan, Carl, 1934-
     (END)
```

CHOICE: R

Suppose R3, The Cosmic Connection, is book you want. Type 3 and <CR>: the record appears in a form similar to a catalog card. This record is reproduced on the next page with tags to show what each section of the display means.

Sample Record

```
523 S12c                        <=CALL NUMBER

  Sagan, Carl, 1934-           <=AUTHOR

     The cosmic connection: an extraterrestrial perspective.  Produced by
Jerome Agel.
                        ^ TITLE

     Garden City, N.Y., Anchor Press, 1973.  <=PUBLISHER,DATE

     Life on other planets.                  <=TRACINGS

     Life-Origin.

LOCATION          CALL#/VOL/NO/COPY           STATUS

E&S               523    S12c c.1            available
```

The location is Engineering & Science Library, there is one copy of this book (Vol and No are for magazines,) and it's available--that is it is not checked out.

USER PROTOCOL OF EXAMPLE:

O.k., I'm ready to try my first library search. I've got my nifty little instruction book here so I'm ready to go. I'm interested in finding out about the most recent research on artificial intelligence. I heard that Allen Newell has written a kind of state of the art paper, but I don't know what the name of it is. I hope this system will help me find it and other interesting articles. Let's see now, ILS: Integrated Library System. What is ILS?

The Integrated Library System, known as ILS, provides access to a catalog of holdings of all branches of the University Libraries. The system is available to you from terminals placed in the libraries and currently contains about 8,000 test records. *Blah, blah who cares?*

ILS is designed to be used by new users who are not familiar with online catalogs. As a user you won't have to worry about destroying any part of the system or "breaking something". The system is designed to only DISPLAY any information you request of it. *o.k., let's see what it does.*

ILS will PROMPT you as to the next step in your search and will tell you when you've done something wrong. It will also suggest ways to help you fix your mistakes. As you progress in searching ILS you will be able to shortcut these step by step directions and use the short commands on pages 10 and 11. If you need help you can type HELP followed by CARRIAGE RETURN; ILS will respond with instructions. *Fine, now what do I do? How do I start this thing?*

Getting Started and Sample Search. When you sit down at the TERMINAL it will either be displaying the first screen of ILS or part of the last person's search. You will need to start at the first screen. To do this, type /ES until the screen contains...*All right, type slash ES...here we go. No, here we don't go...Nothing. Well, how about a carriage return? O.k., there it is.*

Choose the type of Search You Wish to Perform: 1 - BY AUTHOR. 2 - BY TITLE. 3 - CORPORATE/CONFERENCE. 4 - OTHER. OR ENTER A TERM. *Well, I'm not so sure which of these I want. I wonder if you get the same information through any of these searches if you type the same author? I guess I want to search by title. I wonder if this title means books and articles? Let's see what they say. Oh, this stuff below is just for books. I don't want a book, so this won't help. I think I'll* *just skip this part. Oh no, Carl Sagan. I never could stand him. Well, guess I'll try by title--o.k., hit two. Now they want the title--History of Artificial intelligence. That's about all I know. It's not responding, now what? I guess that's not the right title. I thought it would tell me what the title is. I don't even know how to spell the name. Let's try N-e-w-e-l. No, that's not right. How do I delete? Shit. This is not working. How do I get back to the first screen? How do I get help? I forgot. Well I'll just type help and see what happens. H-E-L-P...O.k., options...blah blah, just get me back to the beginning. There it is. Now I'm back at the top. Let's see if the Carl Sagan stuff will help any.*

Suppose you wanted to find out if there are any books in ILS by Carl Sagan. You would type 1 (the number of a search by author) and <CR>. ILS will respond with a screen that has three spaces in which to type the author's name. Notice that the CURSOR is positioned at the beginning of the space where you are to enter information. Type the author's last name (Sagan) and a <CR>; the cursor moves to the next information block. Type the author's first name (Carl) and a <CR> (If you don't know the author's first name simply type <CR> here); the cursor moves to the beginning of the third information block. We don't know Carl Sagan's middle name, so just type <CR> ILS will respond with a screen that contains REFERENCE NUMBERS, the number of items that contain Carl Sagan as author, and the name of the author as it appears on the RECORD. *I'm afraid to try this stuff because I may never get back out. Oh well, be brave. Carl baby would be proud of me. Typing 1...Sagan, return. Type Carl, hit return. O.k., that should do it.* At the bottom of the screen is CHOICE:R, Author: Sagan, Carl REF.TITLES.AUTHOR. R1 5 SAGAN, CARL, 1934-.

Now wait a minute, that's not what's on my screen at all. What is this? I should have known better. Why doesn't this match what I see on my screen? Well, I can't see how I'm ever going to find information on what I want if I can't even replicate the example. I am wasting my time, I should be out of here by now. I don't even know how to log out. Too bad, it's not my account anyway. I'm leaving. Sorry if I messed up this taping.

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- How do I find out about recent research on artificial intelligence?
- Will the system help me to find Allen Newell's article along with other related articles?
- What does ILS do?
- How do I start the search?
- Do I hit a carriage return after typing slash ES?
- How do I know which search I want? Will any search produce the same result if given the same author?
- I wonder if searching for articles can be done with the book search?
- How do I get help if the system does not respond?
- How do I delete?
- How do I get back to the first screen?
- Why doesn't the example match what I see on my screen?
- How do I log out of ILS?

A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

WHAT IS ILS?

The Integrated Library System, known as ILS, provides access to the holdings of all branches of the University Libraries. It contains records for books, journals, government documents, technical reports, musical scores and records.

How to use ILS

Using the Menu to Start a Search

ILS is designed to be used by new users who are not familiar with online catalogs. It gives a menu of the most frequently used searches on the top screen of the system. When you type /ES or hit the NEW SEARCH function key, ILS begins by showing you all the types of searches you can do. You can use that menu to help you choose a search.

Using Commands to Start a Search

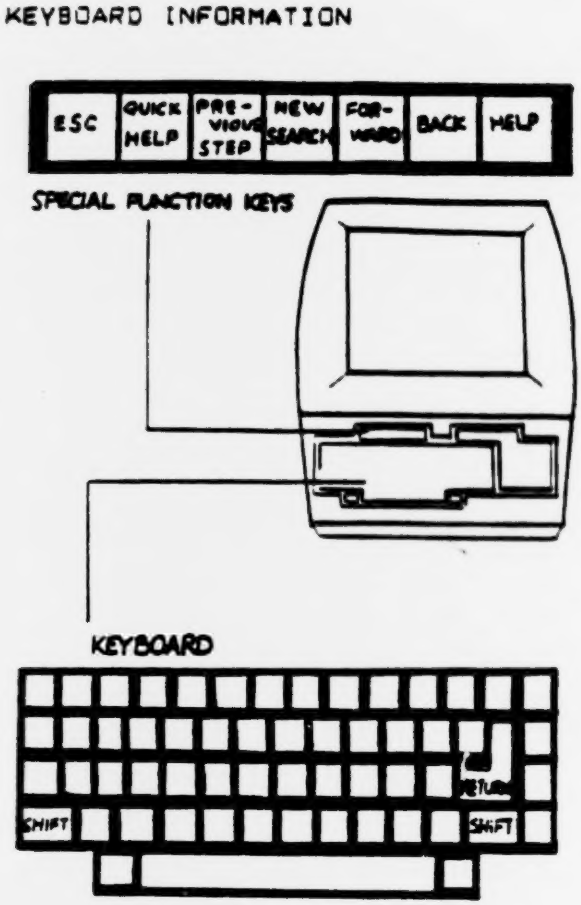
ILS is also designed to be searched using the commands on page 10. Users who are familiar with searching other on-line catalogs may want to shortcut the menus and start searches using commands.

How to use this guide

The purpose of this guide is to help you to learn how to search ILS and to serve as a quick reference quide. Within this brochure you will find:

- How to use ILS
- How ILS' keyboard looks
- How to start a search
- How to decide what kind of search to do
- How to search for an author
- How to search for a title
- How to search for a subject using one term
- How to narrow a search
- A list of abbreviations used in the catalog

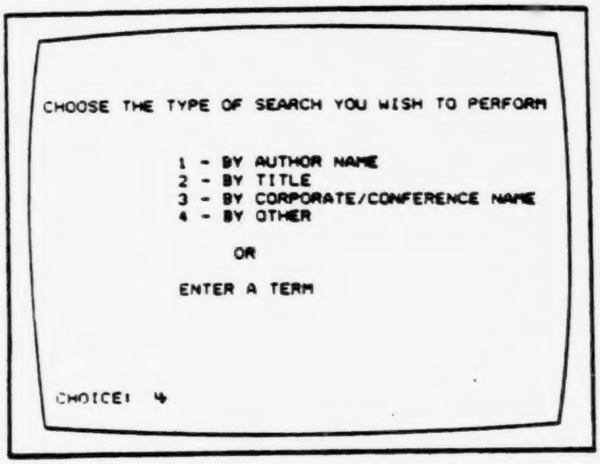
Before you start, compare your ILS keyboard with the diagram below. Notice that the ILS keyboard has special function keys as well as regular alphanumeric keys:


KEYBOARD INFORMATION
ESC
QUICK HELP
PRE-VIOUS STEP
NEW SEARCH
FOR-WARD
BACK
HELP
SPECIAL FUNCTION KEYS
KEYBOARD
RETURN
SHIFT
SHIFT

GETTING STARTED

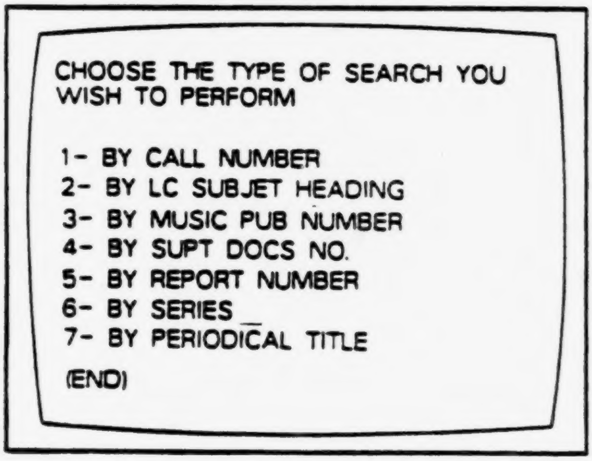
HOW TO START A SEARCH

To START OR END a search hit the **NEW SEARCH** function key or type /**ES** and hit the **RETURN** key. You will see a screen that looks like this:



```
CHOOSE THE TYPE OF SEARCH YOU WISH TO PERFORM

                  1 - BY AUTHOR NAME
                  2 - BY TITLE
                  3 - BY CORPORATE/CONFERENCE NAME
                  4 - BY OTHER

                        OR

                  ENTER A TERM


CHOICE: 4
```

You will notice that the initial screen gives you 5 options (Author, Title, Corporate/Conference Name, Other, and Term). In addition to these types of searches, you can go to a second menu of search types using the Other option (by typing **4** and hitting **RETURN**). The Other Searches screen will then give you these choices:



```
CHOOSE THE TYPE OF SEARCH YOU
WISH TO PERFORM

1- BY CALL NUMBER
2- BY LC SUBJET HEADING
3- BY MUSIC PUB NUMBER
4- BY SUPT DOCS NO.
5- BY REPORT NUMBER
6- BY SERIES
7- BY PERIODICAL TITLE
(END)
```

This second screen of search types offers 9 additional searches (Call Number, LC Subject Heading, Music Publishing Number, Superintendent of Documents Number, Report Number, Series, Periodical Title, Language, and Author/Relator for Fine and Rare Books). These searches are explained on page 12.

HOW TO CHOOSE A SEARCH

ILS catalogs the information about a work in several ways. If, for example, you want to find D.N. Perkins' *The Mind's Best Work*, an AUTHOR SEARCH or a TITLE SEARCH will locate the reference. In this case the search is simple enough that either method works well. But, if you want to find "Protocol Research" and choose a LCSUBJECT HEADING SEARCH for "Protocol," you won't find the information very quickly ("Protocol" isn't a LC Subject Heading, and you'll have to get at the information through "Psychology-Methods"). A TERM SEARCH for "Protocol" however, will locate the relevant titles with ease.

Thus, choosing the best search to perform is a key to searching ILS quickly. To choose an appropriate search you need to

1. Decide which method will produce the information quickly.
2. Be ready to try another search if yours doesn't work.

If You Know	And Want To Find	We Recommend
title	book	title search {Menu choice (2) or /TI}
author	all books	author search {Menu choice (1) or /AU}
general topic	some books	term search {Menu choice (enter a term) or /TM}
specific topic or lc subject headings	some books	term search {Menu choice (enter a term) or /TM} {Menu choice (4) on first menu, then (2) on the second menu or /SU}
conference report	conference name	corporate/conference {Menu choice (3)}

HOW TO SEARCH FOR AN AUTHOR:

TO START:

Choose **1** on the first screen (or type **AU** anywhere in the system) to initiate an AUTHOR SEARCH.

ILS will display a screen that prompts you to type the author's name -- last name first, then first name and then middle initial.

TO ENTER AUTHOR:

When prompted, type the author's last name exactly, and then hit RETURN.

When the cursor is under the first name column, type the author's first name exactly, and then hit RETURN.

When the cursor is under the middle initial column, type the author's middle initial, and then hit RETURN. [NOTE: If you skipped the author's first name or if you don't know the middle initial , just hit RETURN.]

ILS will list the author (or authors) that matches what you typed. If there is no match, ILS will type "NO MATCH THIS ENTRY, RETURN TO CONTINUE."

IF NO MATCH:

Either the libraries do not have works by this author, or the author's name is not cataloged as you typed it.

Have you entered the title in a way that ILS can respond?

ILS does not correct spelling errors. If you are unsure of the spelling of a word, type as much of the author's name as you are sure of and hit RETURN three times.

For Example: If you're looking for works by Andrew Marvell and are unsure of how to spell "Marvell", type **MARVEL** and hit RETURN three times. ILS will give you a list of all the authors whose last names start with that string of letters.

IF SEVERAL MATCHES:

There may be more than one author with the name you typed, or the author you want may have more than one set of records. Be sure to check all the matches if you don't find the particular work you want immediately.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Told the reader the purpose and previewed what to expect in the manual.
- Created screen displays to enable user to see key information needed to start.
- Changed the organization from first presenting a sample author search to how to think about choosing a type of search.
- Created a sample keyboard to clarify distinction between function keys and alpha numeric keys.
- Deleted all sample record information because it was not clear.
- Provided a chart that recommends several types of searches based on what the user may know and what they want to find.

STRATEGIES FOR REVISORS:

- Because user needs are so varied, on-line documentation presents particular problems for writers. On-line documentation users may or may not have a clear idea of what information they want to access. They expect the system to provide choices that will help them to make decsions and to choose between alternative solution paths. This type of user situation places at least three demands on the writer. The writer must:
    - address the hypothetical tasks the user may require of the system
    - construct the context out of which the user can proceed
    - make the user aware of pros and cons of particular solution paths
- Although providing a menu is a good strategy, the user needs to understand the implications of choices they make. This unit's example text, for instance, leads the reader to think that all choices are equal. Notice how the revision focuses on choosing a search rather than on providing sample searches. It also considers what the user may already know that will help to make a wise choice.

UNIT 10

Context: The example text of Unit 10, "**The DUC System**" is selected from the beginning of a tutorial on a computer-aided design systems, "DUC". The user is assumed to have experience neither with "DUC" nor with computing. The revision is aimed as a tutorial for designers who have no familiarity with any computer-aided design system.

EXAMPLE TEXT:

The DUC System

The Display Station

Each display station consists of a terminal with a display screen, a typewriter keyboard, a light pen, and a function keyboard.

The Display Screen

Message Area--provides feedback on the current status, e.g., function currently in use, scale of drawing, warning of invalid operation etc.

Menu Area--provides options which you can choose with the light pen.

The Light Pen

The light pen is a device which serves two functions:

Selecting (Sel)

Selecting an item on the menu area or an element in the drawing area. Hold the pen perpendicular to the screen with the point touching the desired item and then push the pen point into the screen until the pen clicks. The terminal prompts you to select by typing Sel in the message area of the display screen.

Indicating (Ind or Tc)

Indicating an item on the drawing area of the screen. Hold the pen perpendicular to the screen at approximately the desired place and then hit the Indicate function key. The terminal will prompt you to indicate by typing IND or TC in the message area of the display screen.

Both of these functions will be covered in the first exercise.

Throughout this manual, we will designate instructions as follows:

- "Select" always applies to pushing the light pen into the screen until the pen clicks.
- "Indicate" always applies to touching the light pen to the screen at the specified place and hitting the Indicate function.

USER PROTOCOL OF EXAMPLE:

The DUC system. *Well, if I wasn't sitting in front of this terminal, I'd never realize this stuff is about a design system.* The Display Station. Each display station consists of a terminal with a display screen, a typewriter keyboard, a light pen, and a function keyboard. *O.k., so I'm looking at this equipment and trying to tell which is which. I've never used a function box or a light pen before. I want to try out this light pen, it looks fun to use. The only thing is that I'm left handed and I don't know how to position my hands with all this new stuff.*

The Display Screen. *Well I'll assume this means the terminal screen.* Message Area--provides feedback on the current status, e.g., function currently in use, scale of drawing, warning of invalid operation etc. *I really don't know what I should be looking at here. What part is the display screen? I haven't tried anything yet so I don't know why they're telling me this detail now. I don't yet understand current status or invalid operations either.*

Menu Area--provides options which you can choose with the light pen. *All right, but where exactly is the menu area?* The Light Pen. *What is the purpose of the light pen?* The light pen is a device which serves two functions. *O.k., I'm trying to learn the function of this equipment all at once because I'm eager to create my own drawings. I guess for this session, I'll have to be satisfied with the basics. From the introduction, I expect that I'll be moving from drawing points to lines to more complex elements like circles and ellipses. O.k. I'm ready.*

Selecting (Sel). Selecting an item on the menu area or an element in the drawing area. Hold the pen perpendicular to the screen with the point touching the desired item and then push the pen point into the screen until the pen clicks. *I know it says what to do, but I'm having difficulty knowing exactly how to hold this thing. How do you make it perpendicular? I don't know how hard to press and I'm afraid of damaging the screen.* The terminal prompts you to select by typing Sel in the message area of the display screen. *I wonder if it always prompts you?*

Indicating (Ind or Tc). Indicating an item on the drawing area of the screen. Hold the pen perpendicular to the screen at approximately the desired place and then hit the Indicate function key. *Where is the indicate function key located? How can I do both indicating and hitting function keys simultaneously?* The terminal will prompt you to indicate by typing

IND or TC in the message area of the display screen. *What does TC stand for? Where is the message area of the screen? Couldn't they give me a better idea of how to hold the light pen with a drawing instead of words?* Both of these functions will be covered in the first exercise. *What is the essential difference between selecting and indicating? I'm not getting from this description. Why are they telling me these commands without a context?*

Throughout this manual, we will designate instructions as follows. *Is there a reason to tell me the instructions without telling me when and where I will use them?* "Select" always applies to pushing the light pen into the screen until the pen clicks. *Is this really a two-step action, first you select and then you indicate? Or are there times when I simply indicate?* "Indicate" always applies to touching the light pen to the screen at the specified place and hitting the Indicate function. *Sounds fairly clear but I still don't know in what contexts I'd choose these. It seems odd to put them here.*

THE USER'S QUESTIONS WHILE READING THE EXAMPLE TEXT:

- Where should I be looking to get the point of this information?
- Where exactly is the message vs. the menu area?
- What is the purpose of the light pen?
- How do I hold the light pen?
- What is the difference between selecting and indicating?
- Where is the indicate function key located?
- Do I simultaneously hit both indicating and function keys?
- Why are they telling me these commands without a context?
- Is there a reason to tell me the instructions without telling me when and where I will use them?
- Couldn't they give me a better idea of how to hold the light pen with a drawing instead of words?
- What does TC stand for?
- Will the terminal always prompt me for these commands?
- Is this really a two-step action, first you select and then you indicate?

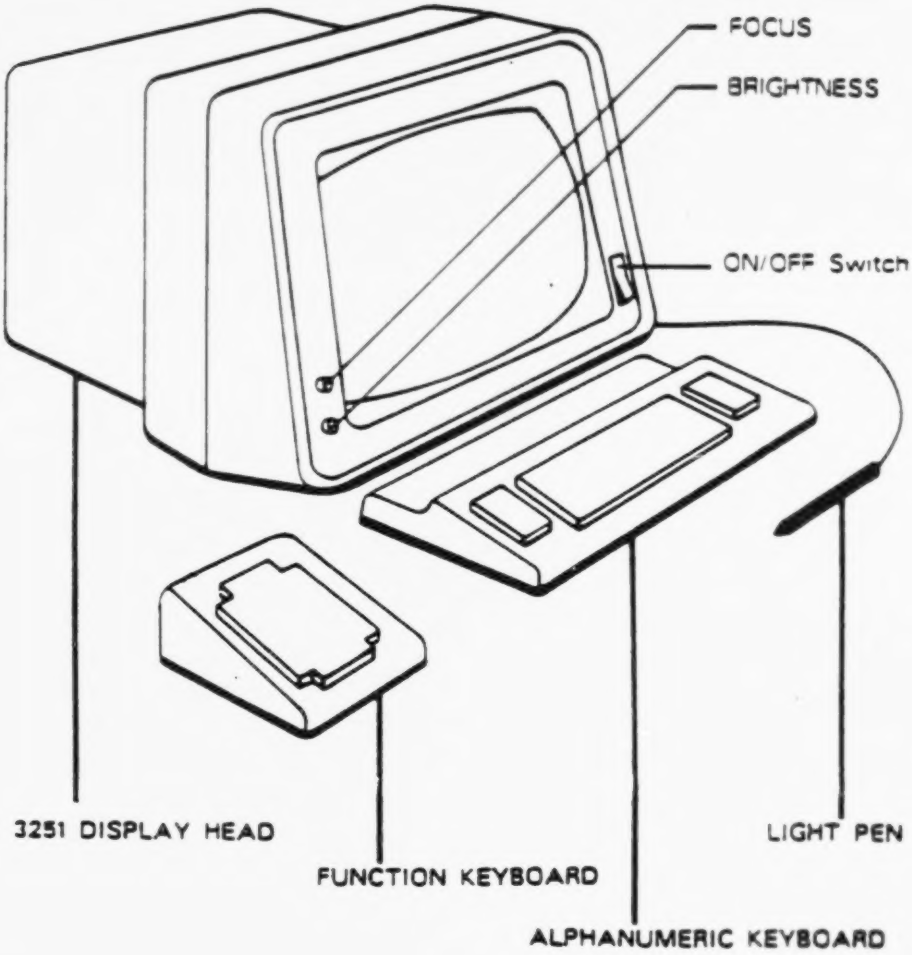
A REVISION BASED ON THE USER'S DIFFICULTIES WITH THE TEXT:

DUC: DESIGN USING COMPUTERS

Equipment

The Display Station

Each display station consists of a terminal with a display screen, a function keyboard, a typewriter keyboard, and a light pen. If you are right handed, it is usually best to hold the light pen in your right hand and position the function keyboard on your left. If you are left handed, do the opposite.

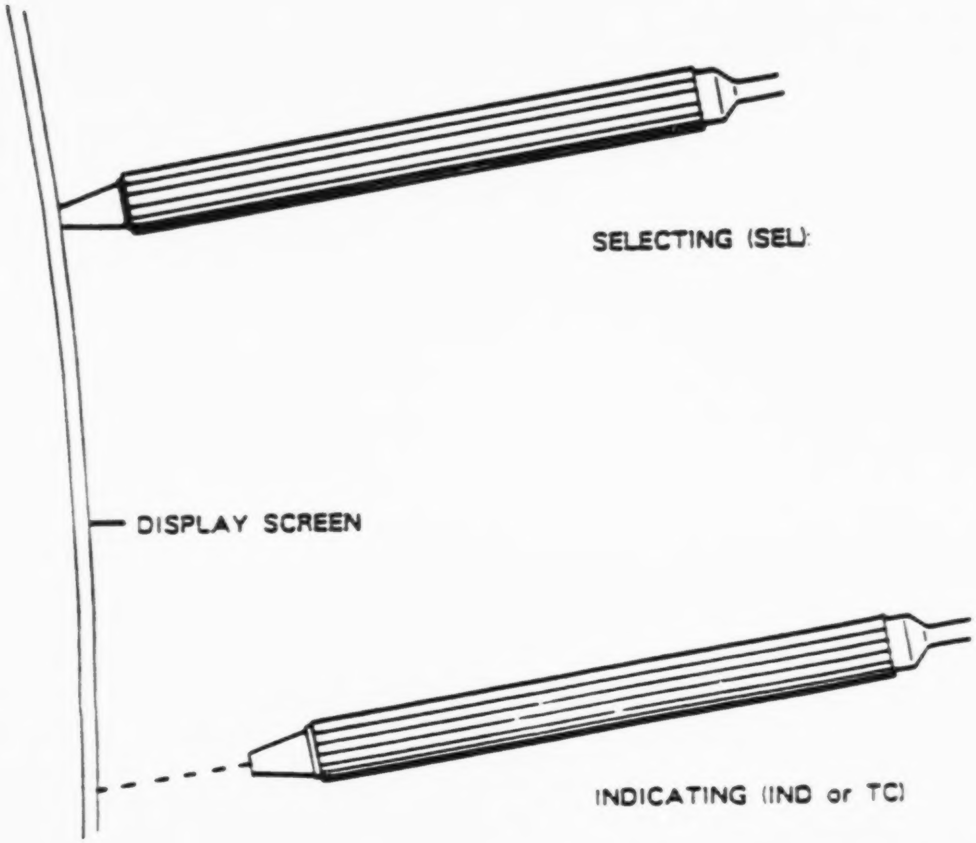

FOCUS
BRIGHTNESS
ON/OFF Switch
3251 DISPLAY HEAD
LIGHT PEN
FUNCTION KEYBOARD
ALPHANUMERIC KEYBOARD

The Light Pen

The light pen serves two functions:

- **selecting** a graphic element (such as a point, line or circle) on the screen or an item on the menu (a list of options on the screen)
- **indicating** an element, an area, or a direction on the screen

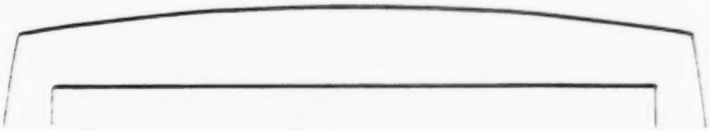
You will learn how to use the light pen in both of these ways on pages 23 and 39 of Exercise One.


SELECTING (SEL)
DISPLAY SCREEN
INDICATING (IND or TC)

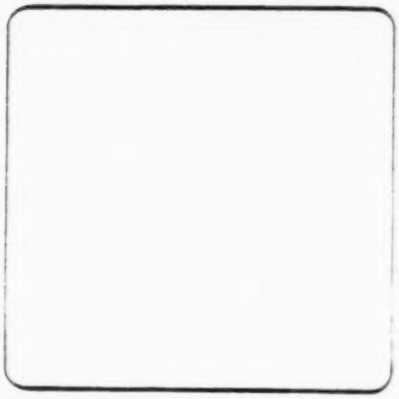
Tips On How To Use This Manual

The symbols below are used throughout this manual to represent areas of the display screen. We describe each area in detail after you login.

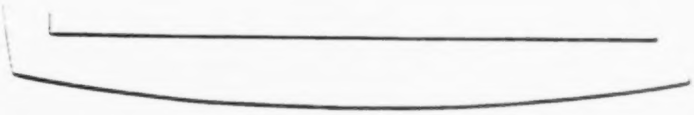
This symbol is used to represent the **message area.**



This symbol is used to represent the **drawing area.**

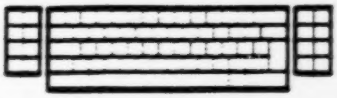


This symbol is used to represent the menu area and the **typing area.**

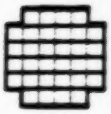


The following symbols are used to represent parts of the **DUC** equipment.

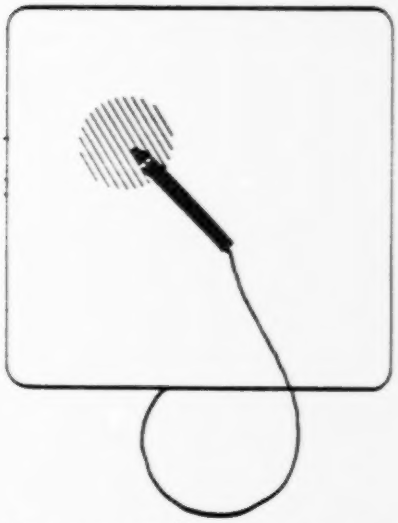
This symbol is used to represent the **typewriter keyboard**. When this symbol appears in the manual, the key you are to hit will be shown in black.



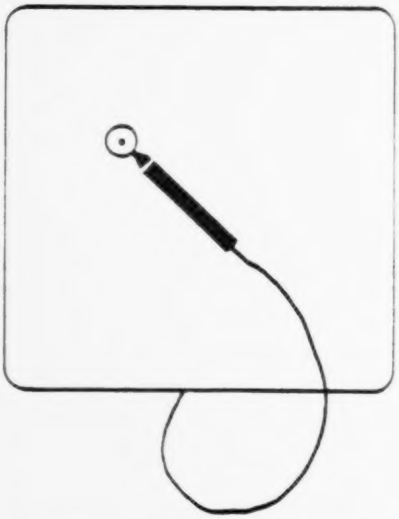
This symbol is used to represent the **function keyboard**. When this symbol appears in the manual, the key you are to hit will be shown in black.



These symbols are used to represent actions with the **light pen**.



This symbol ▨ means that you should **indicate** with the light pen in that approximate area of the display screen.



This symbol ⊙ means that you should **select** with the light pen the element that is shown. The circle is to help you locate the element: it does not appear on the display screen.

STRATEGIES THE WRITER USED TO REVISE THE EXAMPLE:

- Changed the text from a verbal to a graphic presentation.
- Adopted graphic conventions through using visual symbols to remind the reader of the system's special features.
- Presented the material in context.
- Made the page layout easy to follow.
- Presented the information in the order in which it is needed.
- Divided the material into manageable chunks.
- Told the user where they will learn how to use the information.
- Avoided telling the user too much too soon.

STRATEGIES FOR REVISORS:

- Computing situations that require users to shift their focus of attention between various components of equipment, input commands, and output on their terminal screen can be confusing. Novice users often cannot decide what to focus on. When readers have this difficulty, one way to control the reader's attention is through adopting consistent graphic conventions, conventions that call attention to the system's unique features.

- If you are introducing new equipment (e.g., this unit's description of the light pen), it is often best to say it with a visual. Notice how the revision eliminates most of the reader's problems by including how the light pen looks and by showing how to hold it in relation to the display screen.

- When equipment is unfamiliar to users, they need explicit information about relating the equipment's function to particular goals they have for using the system. This unit's revision tells the reader specifically that selecting pertains to points, lines, or circles--while indicating is used for elements, areas, or directions on the screen.

- When planning a document for novice users, it is often necessary to separate functions from procedures, especially when the user must understand groups of functions before invoking procedures. If the document is a tutorial, consider chunking the material into an order through which users ought to proceed. When functions are presented first, new users will better understand the system/program before trying it. It will also signal experienced users to read function information quickly (if at all) and will allow them to move on to the procedures. Notice how the revised version of the light pen information first specifies its basic functions and then tells the reader they will learn to use the pen later in the first exercise. Since users need to know information about other system functions and procedures, such as logging on and using the function keyboard, *before* they can actually begin to draw, the revisor separated the material accordingly.